Module Suite 3.9.0 User Manual

AnswerModules Sagl
AnswerModules Sagl

Module Suite 3.9.0 User Manual

About this guide

Release Notes

Architecture

Installation and Upgrade

Administration

Content Script

Getting started 244

Smart Pages

Script Console

Working with Script Console 572

Extension for DocuSign

Working with DocuSign 581

How to

Content Script: Retrive information 586

About this guide

Audience and objective¶

Module Suite is a collection of solutions that extend the capabilities of OpenText Content Suite and can be successfully deployed to cover a wide range of tasks, from very simple automation operations to more complex and complete applications.

This guide is structured to target those who intend to create, deploy, use, and maintain applications using Content Script, Beautiful WebForms or Smart Pages, and/or want to have a deeper understanding of the possibilities and what can be achieved with the solutions. It is also intended to help the administrators of systems that deploy Module Suite Components.

Prerequisites¶

The majority of this manual has been designed to be accessible to anyone familiar with the basic end-user features of OpenText Content Server. Readers are expected to be comfortable with creating items, navigating workspaces and searching for items. Although not essential, the following knowledge is beneficial:

- OpenText Content Server Knowledge Fundamentals
- Familiarity with the basics of HTML
- Ability to create simple LiveReports or WebReports
- Knowledge of the DTree view from the OpenText Content Suite schema

Release Notes

Version 3.9.0 - Release notes¶

Release Date	End of AMP(*)	End of Life
2025-09-18	2028-09-18	2029-09-18

(*) Active Maintenance Period

The present document contains information regarding product enhancements, fixed issues and known issues related to AnswerModules Modules Suite version 3.9.0.

This guide

The information presented in the on-line guide are mostly non-version specific. AnswerModules team does its best to ensure that, where necessary, is made clear that the information presented is only applicable to specific versions, however if you are looking for this version-specific documentation, you can find it here *(http://developer.answermodules.com/manuals/3.9.0)*

No Warranties and Limitation of Liability

Every effort has been made to ensure the accuracy of the features and techniques presented in this publication. However, AnswerModules accepts no responsibility and offer no warranty whether expressed or implied, for the accuracy of this publication.

Module Suite Compatibility Matrix¶

OpenText Content Server	MS 3.4.0	MS 3.5.0	MS 3.6.0	MS 3.7.0	MS 3.8.0	MS 3.9.0
Content Suite 21.1	X					
Content Suite 21.2	X					
Content Suite 21.3	X					
Content Suite 21.4	X					
Content Suite 22.1	X	X	X	X	X	
Content Suite 22.2	X	X	X	X	X	
Content Suite 22.3	X	X	X	X	X	
Content Suite 22.4	X	X	X	X	X	
Content Suite 23.1	X(*)	X	X	X	X	

OpenText Content Server	MS 3.4.0	MS 3.5.0	MS 3.6.0	MS 3.7.0	MS 3.8.0	MS 3.9.0
Content Suite 23.2		X	X	X	X	X
Content Suite 23.3		X	X	X	X	X
Content Suite 23.4			X	X	X	X
Content Suite 24.1			X(**)	X	X	X
Content Suite 24.2				X	X	X
Content Suite 24.3				X(***)	X	X
Content Suite 24.4					X	X
Content Suite 25.1					X(****)	X
Content Suite 25.2					X(*****)	X
Content Suite 25.3						X

(*) Requires hotfix hotFix_ANS_340_010 to be installed

(**) Requires hotfix hotFix_ANS_360_009 to be installed

(***) Requires hotfix hotFix_ANS_370_003 to be installed

(****) Requires hotfix hotFix_ANS_380_007 to be installed

(*****) Requires hotfix hotFix_ANS_380_013 to be installed

Major Changes in version 3.9.0¶

- Added German, French, and Italian support to Module Suite IDEs
- Performance optimizations and internal refactoring of sandbox isolation layer
- Significant improvements to autocompletion feature with AI-based completion
- Cache service now supports persisting data (as serialized in string) on the database
- ProcessBuilder API now supports blocking attachments as work packages and resetting roles in createDefinition()
- WorkflowForm retrieved from the workflow work package now supports updates via map assignment (task.forms.Form << aMap)
- You can now use println() in scripts, in addition to "out"
- Removed dependencies on helper CSServices (amsuihelper, cshelper, getLog, bwfhelper) - now included in product core

Large Language Models (LLM) Integration¶

- Introduced comprehensive support for OpenAI's Batch API for cost-effective, asynchronous processing

- Implemented service wrappers for OpenAI APIs including file management, evaluations (evals), and fine-tuning
- Enhanced CARL (Chat) widget integration in both Smart Pages IDE and Beautiful WebForms IDE
- Updated CARL chat widget to use Synchfusion widget, manage attachments, and support Response API
- For detailed information on LLM integration, see LLM Integration Guide
- For OpenAI Batch Processing, Evaluations, and Fine-Tuning, see OpenAI APIs Guide

Model Context Protocol (MCP) Integration¶

- Introduced support for Model Context Protocol (MCP) integration
- Enables AI models to access external tools and resources through a secure, capability-based negotiation system
- Supports OAuth2 and custom authorization mechanisms
- Provides seamless integration with OpenAI function calling
- For detailed information on MCP integration, see MCP Integration Guide

SmartUI Commands¶

- Introduced handlers for common SmartUI actions
- Enhanced Smart Pages widgets with new capabilities:
    - Button widget can run actions without refreshing the view, shows a loader, and publishes results to specified channels
    - QueryBuilder widget now supports an explicit "Filter" button
    - New widget for managing maps
- Support for triggering CLEH actions via UI elements and publishing results on ampagenotify channel
- For comprehensive information on SmartUI commands, see SmartUI Commands Guide

Beautiful WebForms¶

- **AI-Based Form Builder**: Introduced an experimental feature that enables designers to create forms using natural language prompts. Instead of manually dragging and dropping widgets, you can simply describe the form requirements, and C.A.R.L. (the AI assistant) will automatically generate the form for you. The feature uses an agentic workflow architecture with a coordinator agent and specialized widget agents working in parallel. You can also use it to configure individual widgets without modifying the rest of the view. All AI-generated changes require explicit user confirmation before being applied. For detailed information, see AI-Based Form Builder
- Enhanced SmartView Task view template with major optimizations
- Support for UI elements triggering CLEH actions and event-based notifications via ampagenotify

- Updated SmartView Task Config widget to support auto-creation of confirmation modal linked to a template's button
- Allow client-side validation to target specific inputs via 'validation' data attribute with comma-separated field IDs
- New form widget for managing maps
- Modified the way SmartUI resources are loaded in SmartView Task view Template

Adobe Sign Integration¶

- Introduced Module Suite Extension for Adobe Sign
- Comprehensive API for creating and managing signing agreements
- Support for uploading documents, managing participants, and tracking agreement status
- Webhook support for real-time notifications
- OAuth 2.0 authentication with automatic token management
- For detailed information on Adobe Sign integration, see Adobe Sign Integration Guide

Content Script¶

- **AI Autocompletion**: Introduced intelligent, context-aware code suggestions directly within the Script Editor. The feature analyzes both the full set of available Content Script APIs and the current context of the script being edited to generate relevant completions. Includes two configurable options:
  - **Smart Completion**: Enables AI-driven code suggestions (requires C.A.R.L. integration to be properly configured and enabled)
  - **Predictive Completion**: Automatically precomputes possible code completions in the background, storing them in a cache for enhanced responsiveness
- New "htmlToText" API in Html extension package
- Removed dependencies on cshelper and getLog CSServices (now included in product core)
- For detailed information on AI Autocompletion, see Script Editor - AI Autocompletion

Administration¶

- **Seal Content Script Versions**: New administrative setting that blocks the creation of new versions of Content Scripts by users with standard permissions. Only administrators and those explicitly allowed to create scripts can modify versions, helping to ensure the integrity of your productive environments
- **Limit Administrators**: New administrative feature recommended on productive systems. When activated, it restricts System Administration users from creating or updating Content Scripts unless they are members of the Privilege group
- For detailed information on these administrative features, see Module Suite Administration Tools

All Enhancements in version 3.9.0¶

ID	Scope	Description
#002191	Smart Pages	Update CARL chat widget in order to use Synchfusion widget, manage attachments and support Response API
#002190	Beautiful Webforms	Update CARL chat widget in order to use Synchfusion widget, manage attachments and support Response API
#002151	Extension - LLM	OpenAI's Batch API Support
#002163	Extension - LLM	Implementation of different service wrappers for OpenAI APIs (file, eval, finetuning)
#002188	Smart Pages	Introduced handlers for common SmartUI actions
#002165	Smart Pages	Integrate CARL (Chat) in SmartPage IDE
#002079	Module Suite	Added German, French, and Italian support to Module Suite IDEs
#002157	Beautiful Webforms	Integrate CARL (Chat) in BWF IDE
#002137	Core	Improve objects and methods descriptions
#002146	Extension - Cache	It is now possible to persist data (as serialized in string) on the database
#002187	Module Suite	You can now use println() in scripts, in addition to "out"
#002186	Module Suite	ProcessBuilder API now supports blocking attachments as work packages and resetting roles in createDefinition()
#002185	Module Suite	WorkflowForm retrieved from the workflow work package now supports updates via map assignment (task.forms.Form << aMap)
#002182	Module Suite	Updated dependencies for the PDF Viewer Template (PDF Viewer Tool)
#002178	Smart Pages	Button can run actions without refreshing the view, shows a loader, and publishes the result to a specified channel
#002177	Smart Pages	Added an explicit "Filter" button to the QueryBuilder widget
#002160	Smart Pages	New Smart Pages widget for managing maps
#002129	Content Script	Significant improvements to autocompletion feature + AI based completion
#002069	Beautiful Webforms	Support for UI elements triggering CLEH actions and event-based notifications via ampagenotify
#002065	Smart Pages	It's now possible to add a "filter" button on the Query Builder

ID	Scope	Description
#002062	Beautiful Webforms	Updated SmartView Task Config widget to support auto-creation of confirmation modal linked to a template's button
#002029	Extension - Html	New "htmlToText" API
#002078	Module Suite	Performance optimizations and internal refactoring of sandbox isolation layer
#002077	Smart Pages	Remove dependency on amsuihelper CSService (now included in product core)
#002076	Content Script	Remove dependency on cshelper, getLog CSService (now included in product core)
#002075	Beautiful Webforms	Remove dependency on bwfhelper CSService (now included in product core)
#002071	Smart Pages	Trigger CLEH actions via UI elements and publish results on ampagenotify channel
#002070	Beautiful Webforms	Support for triggering CLEH actions via bwf:{formid}:action event (ampagenotify) with enhanced execution control
#002068	Beautiful Webforms	Allow client-side validation to target specific inputs via 'validation' data attribute with comma-separated field IDs
#002066	Beautiful Webforms	Major changes in "smartuiwftask" CSService to support changes in SmartView Task view template (see 2064)
#002064	Beautiful Webforms	Major optimization to SmartView Task view template
#002052	Beautiful Webforms	Modified the way SmartUI resource are loaded in SmartView Task view Template
#002053	Beautiful Webforms	Minor optimizations to SmartView Task view template
#002159	Beautiful Webforms	New form widget for managing maps

Issues Resolved in version 3.9.0¶

ID	Scope	Description
#002050	Smart Pages	The tiles marked as "non-contextual" are in any case reloaded when you navigate, even if the perspective remains the same
#002184	Module Suite	Malformed query in ModuleSuite Report

ID	Scope	Description
#002183	Module Suite	In ContentScript Engine, request parameters were sometimes incorrectly converted into arrays
#002181	Module Suite	Application Builder (custom for Builder) was not using deep merge
#002180	Smart Pages	Fixed an issue in the SmartView template that caused resources to load incorrectly
#002176	Smart Pages	SmartView widget occasionally generates non-compilable code in the Controller
#002152	Extension - LLM	Bug fixes llm service (OPENAI Responses API)
#002147	Module Suite	Bug fixes llm service
#002144	Module Suite	Rotating the Mainlog file might cause an error
#002039	Module Suite	Configuring the user session duration to be dependant on the last login causes issues on the DA
#002067	Content Script	Minor issue on "getLog" CSService

Version 3.8.0 (Venus)- Release notes¶

Release Date	End of AMP(*)	End of Life
2025-01-6	2028-01-6	2029-01-6

(*) Active Maintenance Period

The present document contains information regarding product enhancements, fixed issues and known issues related to AnswerModules Modules Suite version 3.8.0.

This guide

The information presented in the on-line guide are mostly non-version specific. AnswerModules team does its best to ensure that, where necessary, is made clear that the information presented is only applicable to specific versions, however if you are looking for this version-specific documentation, you can find it here *(http://developer.answermodules.com/manuals/3.8.0)*

No Warranties and Limitation of Liability

Every effort has been made to ensure the accuracy of the features and techniques presented in this publication. However, AnswerModules accepts no responsibility and offer no warranty whether expressed or implied, for the accuracy of this publication.

Module Suite Compatibility Matrix¶

OpenText Content Server	MS 3.3.0	MS 3.4.0	MS 3.5.0	MS 3.6.0	MS 3.7.0	MS 3.8.0
Content Suite 21.1	X	X				
Content Suite 21.2	X	X				
Content Suite 21.3	X	X				
Content Suite 21.4	X	X				
Content Suite 22.1	X	X		X	X	X
Content Suite 22.2	X	X		X	X	X
Content Suite 22.3	X	X		X	X	X
Content Suite 22.4		X		X	X	X
Content Suite 23.1		X(*)		X	X	X
Content Suite 23.2			X	X	X	X
Content Suite 23.3			X	X	X	X
Content Suite 23.4				X	X	X
Content Suite 24.1				X(**)	X	X
Content Suite 24.2					X	X
Content Suite 24.3					X(***)	X
Content Suite 24.4						X
Content Suite 25.1						X(****)
Content Suite 25.2						X(*****)

(*) Requires hotfix hotFix_ANS_340_010 to be installed

(**) Requires hotfix hotFix_ANS_360_009 to be installed

(***) Requires hotfix hotFix_ANS_370_003 to be installed

(****) Requires hotfix hotFix_ANS_380_007 to be installed

(*****) Requires hotfix hotFix_ANS_380_013 to be installed

All Enhancements in version 3.8.0¶

ID	Scope	Description
#002001	Extension - Retrofit	Introduced a new extension package, "Retrofit," as a Content Script wrapper for the Retrofit library.
#002000	Extension - PDF	Updated PDFBox dependency to version 3.0.3 for compatibility and performance improvements.
#001999	Content Script	Introduced getActivities API in CSWorkflowInstance to retrieve the complete activity history for a workflow instance.
#001945	Module Suite	CSSearchResult does not return version
#001997	Beautiful Webforms	Allow users to add an initial comment when initiating a workflow from a form.
#001045	Module Suite	Include details about the limits enforced by the installed activation key in the Base Configuration.
#001977	Module Suite	Custom Form Builder now available outside Application Builder

Issues Resolved in version 3.8.0¶

ID	Scope	Description
#002025	Module Suite	Absence of the form attributes causing form load issues (Backport)
#001959	Module Suite	Absence of the form attributes causing form load issues
#001940	Extension - LLM	Configuration regression. Issue with profiles
#002016	Module Suite	Log records with special characters are displayed incorrectly in Script Editor (backport)
#001991	Module Suite	Log records with special characters are displayed incorrectly in Script Editor
#002015	Module Suite	Blazon extension for Content Script does not work (backport)
#001975	Module Suite	Blazon extension for Content Script does not work
#002014	Module Suite	Unable to submit form in case of form row duplicate into the Database (backport)
#001994	Extension - Docx	Issue with html field into docx document
#001967	Module Suite	Select View Params Widget Showing Error
#001954	Module Suite	Workflow getComments API does not return initial workflow comment

ID	Scope	Description
#001951	Extension - Docx	Track Changes comments in a docx show errors when retrieved via the listComments api
#001984	Module Suite	JDBC ignores profile settings for internal otcs profile. Missing database drivers may impact performance on certain env
#001955	Module Suite	Wikipage subtype is -1
#001887	Module Suite	Feature request. ADd new methods purge and restore, as in REST API
#001968	Module Suite	Unable to submit form in case of form row duplicate into the Database
#001998	Smart Pages	Disabling SmartView cache has no effect on commands when the Content Script Volume cache remains enabled.
#001958	Module Suite	Memcache errors while custom templates enabled
#001996	Beautiful Webforms	The submission of a BWF in a workflow fails in xECM 24.4
#001960	Module Suite	ADN ID once generated does not record correct Quantity after Form reload
#001344	Beautiful Webforms	Partial Label - missing for attribute error
#001834	Beautiful Webforms	Smart Dropdown is not handling values correctly when they contains a comma
#001933	Module Suite	Tile Content Script Nodes Table issue
#001964	Module Suite	OT patch is breaking Beautiful Webform display
#001932	Module Suite	xECM SPI method GetBusinessObjectQueryFormBulk is not executed
#001961	Module Suite	Performance degradation detected on SmartUI when modules are not configured
#001928	Module Suite	SmartDropDown widget problems
#001772	Module Suite	Missing docman methods in the online help
#001577	Content Script	After upgrade to 3.3, it is not possible edit WorkFlows that have a Content Script Step
#001630	Content Script	Generic error "No such property: csModulePath" is raised downloading cs.log for all the users, excluding Admin
#001868	Module Suite	GroupName method within users service issue
#001978	Module Suite	PDF document generation issues
#001981	Module Suite	Custom column issue

ID	Scope	Description
#001983	Beautiful Webforms	"Column Headers" widget: setting the value to a new row using the context menu modifies the starting row too
#001943	Module Suite	Fixed various issues in the Application Builder
#001950	Module Suite	Error after saving a view created with the Custom Form Builder of the Application Builder
#001937	Module Suite	Typo error when Search on a specific slice

Version 3.7.0 (Earth)- Release notes¶

Release Date	End of AMP(*)	End of Life
2024-07-12	2027-07-12	2028-07-12

(*) Active Maintenance Period

The present document contains information regarding product enhancements, fixed issues and known issues related to AnswerModules Modules Suite version 3.7.0.

This guide

The information presented in the on-line guide are mostly non-version specific. AnswerModules team does its best to ensure that, where necessary, is made clear that the information presented is only applicable to specific versions, however if you are looking for this version-specific documentation, you can find it here *(http://developer.answermodules.com/manuals/3.7.0)*

No Warranties and Limitation of Liability

Every effort has been made to ensure the accuracy of the features and techniques presented in this publication. However, AnswerModules accepts no responsibility and offer no warranty whether expressed or implied, for the accuracy of this publication.

Module Suite Compatibility Matrix¶

OpenText Content Server	MS 3.2.1	MS 3.3.0	MS 3.4.0	MS 3.5.0	MS 3.6.0	MS 3.7.0
Content Suite 21.1	X	X	X			
Content Suite 21.2	X	X	X			
Content Suite 21.3	X	X	X			
Content Suite 21.4	X	X	X			
Content Suite 22.1	X	X	X		X	X

OpenText Content Server	MS 3.2.1	MS 3.3.0	MS 3.4.0	MS 3.5.0	MS 3.6.0	MS 3.7.0
Content Suite 22.2	X	X	X		X	X
Content Suite 22.3		X	X		X	X
Content Suite 22.4			X		X	X
Content Suite 23.1			X(*)		X	X
Content Suite 23.2				X	X	X
Content Suite 23.3				X	X	X
Content Suite 23.4					X	X
Content Suite 24.1					X(**)	X
Content Suite 24.2						X
Content Suite 24.3						X(***)

(*) Requires hotfix hotFix_ANS_340_010 to be installed

(**) Requires hotfix hotFix_ANS_360_009 to be installed

(***) Requires hotfix hotFix_ANS_370_003 to be installed

New Feature: Improved Support for Long Identifiers

Module Suite 3.7.0 introduces enhanced support for very long identifiers. If your environment utilizes long identifiers, you can enable this improved support through the Module Suite Base Configuration.

To enable this feature:

1. Navigate to the Module Suite Base Configuration settings.
2. Locate the option for long identifier support.
3. Enable the feature as needed.

For detailed instructions, refer to our Base Configuration documentation *(/manuals/3.7.0/administration/modulesuite/#base-configuration)*.

SASL Memcache Authentication Support¶

Module Suite 3.7.0 introduces support for SASL memcache authentication. When enabling this feature on OTCS, follow these important steps:

Single Thread Client Configuration

Ensure that the cache is configured to use a single thread client. To do this:

1. Navigate to the Module Suite base configuration.
2. Locate the `amcs.cache.mode.default` property.
3. Set its value to `single`.

Configuration Reload Required

After enabling SASL authentication on OTCS, you must save the Base Configuration to force a configuration reload.

Steps to Enable SASL Memcache Authentication¶

1. Configure the cache to use a single thread client as described above.
2. Enable SASL authentication in your OTCS settings.
3. Save the base configuration to apply the changes.

Module Suite 3.7.0 Breaking Changes¶

Module Suite 3.7.0 it's based on Groovy 4. Groovy 4 builds upon existing features of earlier versions of Groovy. In addition, it incorporates numerous new features and streamlines various legacy aspects of the Groovy codebase.

Major Groovy Version Update

This release includes a significant update from Groovy 3.0.19 to Groovy 4.0.20. This is a major version change that introduces new features, improvements, and breaking changes. Users should carefully review their existing Groovy code and dependencies for compatibility issues. Key points to note:

- Several breaking changes, including removal of the old parser and classic bytecode generation
- New features like switch expressions, sealed types, and records (some incubating)
- Performance improvements, especially for GString
- Changes in JDK requirements (JDK16+ to build, JDK8+ to run)
- Some modules and classes have been removed or relocated

Please refer to the Groovy 4.0 release notes for a comprehensive list of changes and migration guidance.

Note

WARNING: Some features of Groovy 4 are designated as "incubating". Where appropriate, related classes or APIs of these features may be annotated with the @Incubating annotation. Caution should be exercised when using incubating features as the details may change in subsequent versions of Groovy. We don't recommend using incubating features for production systems.

Important naming/structuring changes¶

Maven coordinate change¶

In Groovy 4.0, the groupId of the maven coordinates for Groovy have changed from `org.codehaus.groovy` to `org.apache.groovy`.

Legacy package removal¶

The Java Platform Module System (JPMS) requires that classes in distinct modules have distinct package names (known as the "split packaging requirement"). Groovy has its own "modules" that weren't historically structured according to this requirement.

Groovy 3 provided duplicate versions of numerous classes (in old and new packages) to allow Groovy users to migrate towards the new JPMS compliant package names. See the Groovy 3 release notes for more details. Groovy 4 no longer provides the duplicate legacy classes.

In short, time to stop using `groovy.util.XmlSlurper` and start using `groovy.xml.XmlSlurper`. Similarly, you should now be using `groovy.xml.XmlParser`, `groovy.ant.AntBuilder`, `groovy.test.GroovyTestCase` and the other classes mentioned in the prior mentioned Groovy 3 release notes.

New features¶

Switch expressions¶

Groovy has always had a very powerful switch statement, but there are times when a switch expression would be more convenient.

```
def result = switch(i) {
    case 0 -> 'zero'
    case 1 -> 'one'
    case 2 -> 'two'
    default -> throw new IllegalStateException('unknown number')
}
```

Sealed types¶

Sealed classes, interfaces and traits restrict which other classes or interfaces may extend or implement them. Groovy supports using a sealed keyword or a @Sealed annotation when writing a sealed type.

```
sealed interface Tree<T> {}
@Singleton final class Empty implements Tree {
    String toString() { 'Empty' }
}
@Canonical final class Node<T> implements Tree<T> {
    T value
    Tree<T> left, right
}
```

Records and record-like classes (incubating)¶

Groovy 4 adds support for native records for JDK16+ and also for record-like classes (also known as emulated records) on earlier JDKs.

```
record Cyclist(String firstName, String lastName) { }
```

Built-in type checkers¶

From Groovy 4, we bundle some select type checkers within the optional groovy-typecheckers module, to encourage further use of this feature.

```
@TypeChecked(extensions = 'groovy.typecheckers.RegexChecker')
def whenIs2020Over() {
    def newYearsEve = '2020-12-31'
    def matcher = newYearsEve =~ /(\d{4})-(\d{1,2})-(\d{1,2}/
}
```

GINQ, a.k.a. Groovy-Integrated Query or GQuery (incubating)¶

GQuery supports querying collections in a SQL-like style.

```
from p in persons
leftjoin c in cities on p.city.name == c.name
where c.name == 'Shanghai'
select p.name, c.name as cityName
```

Other improvements¶

- GString performance improvements
- Enhanced Ranges
- Support for decimal fraction literals without a leading zero
- JSR308 improvements (incubating)
- AST transformation priorities

Legacy consolidation¶

- Old parser removal
- Classic bytecode generation removal

JDK requirements¶

Groovy 4.0 requires JDK16+ to build and JDK8 is the minimum version of the JRE that we support. Groovy has been tested on JDK versions 8 through 17.

All Enhancements in version 3.7.0¶

ID	Scope	Description
#001860	Smart Pages	New Sync PDF Viewer Widget
#001858	Smart Pages	New Kanban Widget
#001923	Module Suite	Updated Synchfusion based widgets after having updated the dependency (25.1.35)
#001922	Smart Pages	Added a self-contained method one can use to render programmatically a Smart Page
#001920	Module Suite	Introduction of a new object-based licensing model
#001919	Module Suite	Uniformed the look and feel of ModuleSuite admin csscripts to OT administrative pages
#001918	Content Script	Updated Velocity macro #csresource
#001917	Extension - OAuth	It is now possible to use SYSTEM as the storage policy (it will use System Data under the hood).
#001916	Extension - OAuth	It is now possible to register an OAuth Profile on the fly
#001914	Extension - LLM	Introduced dedicated openai Service, added Langchain4j dependency to support more models
#001913	Extension - JDBC	Enable encryption for connection towards internal MSSQL database
#001912	Extension - Docx	Improved the way comments are extracted from a document
#001911	Module Suite	Removed dependency from groovy-wslite. Bumped soa-model-core dependency to 2.0.1
#001910	Module Suite	Enabled OT Memcache SASL support. Updated dependencies and added support for a new non-pooled memcache client.
#001908	Module Suite	All services in the serviceContext now receive a notification when the ContentScriptManager updates the serviceConfiguration.
#001899	Module Suite	New APIs to set and get ModuleSuite related system data configuration
#001909	Module Suite	Autocompletion now provides correct information about internal service's ContentScriptAPI objects
#001907	Module Suite	Optimization of the process used to retrieve the current version of a node
#001882		

ID	Scope	Description
	Beautiful Webforms	New APIs to manage the extended definition of category and form template
#001852	Beautiful Webforms	Added Group Settings configuration to the Grid widget (Initial grouping)
#001849	Smart Pages	Added Group Settings configuration to the Grid widget (Initial grouping)
#001863	Smart Pages	Enabled context menu integration for the Grid widget
#001862	Beautiful Webforms	Enabled context menu integration for the Grid widget
#001872	Module Suite	Update Handlebars runtime javascript to version 4.7.8
#001873	Module Suite	In Content Script Editor and BWF Smart Editor updated jquery and lodash library (3.7.1, 4.17.21)
#001894	Module Suite	New application 'Form Workflow Dashboard' of the Application Builder tool
#001881	Beautiful Webforms	New 'XENGADN Dropdown' widget to manage the 'ADN table key lookup' field
#001831	Module Suite	Docbuilder: How to justify a paragraph
#001827	Module Suite	Re-import of an existing Template Folder is not supported by the Transport Warehouse
#001718	Beautiful Webforms	It is possible to configure a button in the footer of the 'SmartView Task'(V4) template to open a Modal Container
#001813	Module Suite	Useless call to GetNodeFast to retrive the Version of a Script that has been loaded with getNodesFast method
#001654	Smart Pages	Re-import of an existing SmartPage is not supported by the Transport Warehouse
#001935	Rend	Changed default rendition engine from (wkhtmltopdf which is now deprecated) to rend

Issues Resolved in version 3.7.0¶

ID	Scope	Description
#001896	Module Suite	xECM SPI method GetBusinessObjectQueryFormBulk is not executed
#001759	Module Suite	rhRequest not working
#001883	Module Suite	Modification of public rights issue

ID	Scope	Description
#001761	Module Suite	info.pageCount wrong results on word generation
#001904	Script Console	Script Console sample security configuration does not activate CSRF protection to Remote WebForms extension urls
#001832	Module Suite	SFTP configuration issue
#001816	Module Suite	Business Application using a connector type from xECM for Everything causing trace files
#001843	Module Suite	Setting classification results in error 'could not login with cookie'
#001886	Module Suite	Big integer DataID (= 10000000000) returned incorrectly in Content Script
#001892	Content Script	It is not possible to set "SQL Table" as the storage mechanism of a workflow form via the workflow process builder
#001897	Module Suite	Error when a new version is added to a workflow attachment via the 'Workflow attachments' section of 'SmartView Task' template
#001898	Beautiful Webforms	Minor visualization issues on CARL widget (widget's height non properly set)
#001850	Module Suite	Smart Dropdown widget not working correctly when switch to another tab
#001806	Module Suite	Rest call delay and session expiration
#001871	Module Suite	Workflow update step package instructions API issue
#001369	Module Suite	Issue in the Base Configuration page
#001835	Smart Pages	In some cases odata crud operations return an error
#001879	Module Suite	ADN Reference field implementation (or ADN ID upgrade)
#001867	Beautiful Webforms	The ADN Dropdown widget fails to retrieve ADN table key lookup values
#001895	Module Suite	Space Content Widget issue
#001878	Online Documentation	Flag name for message leads to imap error fetching mail
#001876	Online Documentation	Documents without extension when temporary file crated, dot is added
#001844	Module Suite	Version Content Fails If Document name is not FS compatible
#001853	Module Suite	Anscontentsmartui module during Download create a trace file
#001884	Module Suite	

ID	Scope	Description
		Default value set on BWF widgets overrides current non-empty form field value when loading a previously submitted form
#001784	Module Suite	Using i18n map in Content Script for a locale
#001767	Module Suite	Script Editor does not show snippets after the full Snippets import in MS 3.5 (after Blazon snippet import)
#001837	Module Suite	Field not emptied with Smart DropDown
#001855	Module Suite	Smart Dropdown Validation rises even if it has a value set
#001838	Module Suite	Smart Dropdown Breaking in Set
#001775	Module Suite	Updated dependencies presenting risks related to security vulnerability
#001847	Module Suite	Issue with removing categories from nodes. The script terminates correctly but the categories are not removed.
#001840	Smart Pages	Search time keeps adding on
#001823	Beautiful Webforms	The getFormInfo method loads incorrect information in the definition of fields belonging to a set
#001758	Module Suite	Cache.touch method not working as expected
#001822	Beautiful Webforms	Submitting a Content Server Versions form with unchecked checkboxes using forms.submitForm API will result in invalid data
#001826	Smart Pages	Using the Include Web Form widget, fields with the error are not highlighted
#001812	Module Suite	Fixed various issues in the Application Builder
#001825	Module Suite	AM Logo is not up-to-date in the Velocity macro
#001708	Module Suite	Application Builder: In the 'Document Builder' application, the 'Panel Container Toolbar' widget is not displayed in the form
#001707	Module Suite	Application Builder: It is not possible to create the 'Create and Approve' application
#001790	Content Script	Using the new Extension for Extended ECM for Engineering the generated transmittal 'Load sheet' is a csv instead of an xlsx
#001808	Content Script	The overrides of the anscontentscript module are not loaded in the correct order
#001824	Extension - xECM	When creating a BWS if the attached category has a default Date applied the creation fails
#001755	Module Suite	Typo error in Custom Script Widget

ID	Scope	Description
#001558	Beautiful Webforms	BWF Editor: opening the editor can require more that 10 seconds
#001788	Module Suite	Content Script static variables incorrect Long conversion
#001810	Content Script	The escapeXML method of the html API does not work
#001786	Extension - xECM	The fluent api newBusinessWorkspaceCreationRequest to create the BWS (xecm API) does not create the Transmittal workspace
#001809	Extension - xECM	Run Content Script Action in Event Bots Configuration page can't be properly configured
#001807	Extension - LLM	Error when defining a new function
#001785	Smart Pages	The Grid widget does not pass the parameter to the odata service.
#001789	Beautiful Webforms	The docman.getNode().update() method returns an error when trying to update the Beautiful Form
#001793	Content Script	Module Suite notifications in the 'Notification Center' do not display the header title correctly
#001787	Content Script	The getFacetsVolume() method of the docman api returns the wrong volume
#001163	Online Documentation	Review license pages
#001072	Online Documentation	Requirements page is not updated
#001115	Online Documentation	Installing Extension Packages
#001418	Online Documentation	Add a note to remove CSSystem
#001666	Online Documentation	Dead link in Getting Started page
#001650	Online Documentation	Add deprecation information for wkhtmltopdf rendition method
#001658	Online Documentation	Wrong anchor link in Callback documentation page
#001671	Online Documentation	Module Suite documentation page "Installing on a clustered environment" is incomplete for the reconcile of opentext.ini file
#001673		

ID	Scope	Description
	Online Documentation	Broken link on the "Deploy on Windows" documentation page in SAP extension section
#001783	Online Documentation	Missing extension when renaming war files in paragraph "What to do if the installer raises the error: Unable to automatically.."
#001796	Online Documentation	Impersonation documentation of the API is missing
#001889	Online Documentation	Add mandatory Module Suite Extensions to Deployment guide on Developer Website
#001890	Online Documentation	Enabling OT Memcache SASL requires to save the AnswerModules Base Configuration
#001876	Online Documentation	Documents without extension when temporary file crated, dot is added

Dependencies updated in version 3.7.0¶

Library	Previous Version	New Version
groovy	3.0.19	4.0.20
commons-logging	1.2	1.3.1
commons-validator	1.7	1.8.0
commons-text	1.11.0	1.12.0
commons-email	1.5	1.6.0
commons-net	3.9.0	3.10.0
commons-io	2.13.0	2.16.1
commons-lang3	2.13.0	3.14.0
commons-codec	1.11.0	1.17.0
okhttp	4.11.0	4.12.0
jackson-databind	2.13.5	2.17.1
log4j-api	2.20.0	2.23.1
slf4j-api	2.0.6	2.0.13
handlebars	4.3.1	4.3.1
fop	2.8	2.9
gpars		removed
c3p0	0.9.5.5	0.10.1

Library	Previous Version	New Version
httpclient	4.5.13	4.5.14
http-builder		removed
pdfbox	2.0.26	2.0.31
guava	11.0.1	33.2.0-jre
javaparser-core	0.9.1	1.5.2
jsoniter		removed
aws-java-sdk	1.12.490	1.12.723
woodstox-core	6.5.1	6.6.2

Version 3.6.0 (Genève)- Release notes¶

Release Date	End of AMP(*)	End of Life
2023-11-26	2026-11-26	2027-11-26

(*) Active Maintenance Period

The present document contains information regarding product enhancements, fixed issues and known issues related to AnswerModules Modules Suite version 3.6.0.

This guide

The information presented in the on-line guide are mostly non-version specific. AnswerModules team does its best to ensure that, where necessary, is made clear that the information presented is only applicable to specific versions, however if you are looking for this version-specific documentation, you can find it here *(http://developer.answermodules.com/manuals/3.6.0)*

No Warranties and Limitation of Liability

Every effort has been made to ensure the accuracy of the features and techniques presented in this publication. However, AnswerModules accepts no responsibility and offer no warranty whether expressed or implied, for the accuracy of this publication.

Module Suite Compatibility Matrix¶

OpenText Content Server	MS 3.2.0	MS 3.2.1	MS 3.3.0	MS 3.4.0	MS 3.5.0	MS 3.6.0
Content Suite 21.1	X	X	X	X		
Content Suite 21.2	X	X	X	X		

OpenText Content Server	MS 3.2.0	MS 3.2.1	MS 3.3.0	MS 3.4.0	MS 3.5.0	MS 3.6.0
Content Suite 21.3	X	X	X	X		
Content Suite 21.4	X	X	X	X		
Content Suite 22.1	X	X	X	X		X
Content Suite 22.2		X	X	X		X
Content Suite 22.3			X	X		X
Content Suite 22.4				X		X
Content Suite 23.1				X(*)		X
Content Suite 23.2					X	X
Content Suite 23.3					X	X
Content Suite 23.4						X
Content Suite 24.1						X(**)

(*) Requires hotfix hotFix_ANS_340_010 to be installed

(**) Requires hotfix hotFix_ANS_360_009 to be installed

All Enhancements in version 3.6.0¶

ID	Scope	Description
#001726	Content Script	xECM for Everything - Internal paging details are currently not passed in "listBusinessObjectsWithFilters" function
#001070	Online Documentation	[Documentation] All the links in Packages page are broken
#001738	Smart Pages	SmartPage Widgets are now loaded from the entire volume (as for BWF Widgets)
#001654	Smart Pages	Re-import of an existing SmartPage is not supported by the Transport Warehouse
#001709	Smart Pages	SmartView Actions scripts are invoked twice when the nodes metadata page is displayed
#001714	Module Suite	Flatpickr widget - czech language
#001740	Module Suite	CARL Tool (widgets, and llm service) update
#001751	Module Suite	Application Builder Update: Significant Server-Side Form Builder Enhancements
#001750	Beautiful Webforms	Enhancement: Full Path Specification for Script Snippets in BWF Widgets

ID	Scope	Description
#001749	Smart Pages	Enhancement: Full Path Specification for Script Snippets in SmartPage Widgets
#001748	Beautiful Webforms	New Widget Introduction: Spreadsheet for Enhanced User Experience
#001730	Beautiful Webforms	Improved usability on FormBuilder
#001747	Beautiful Webforms	Enhancement of Server-Side Rendering Support with layoutItems Variable in Handlebars Widget Templates
#001107	Online Documentation	Java version required for the Script Console
#001086	Online Documentation	Script Console configuration page: specify better the port
#001081	Online Documentation	Little change in Event/Callback page

Issues Resolved in version 3.6.0¶

ID	Scope	Description
#001727	Module Suite	Issue with attachment file name UTF-8
#001647	Module Suite	Enhance CSWS API documentation: describe the new methods and provide examples on how replace deprecated ones
#001705	Module Suite	It is not possible to update a BWF via admin.importXml API or via Transport Warehouse.
#001262	Online Documentation	Missing page with requrements for the Script Console
#001257	Online Documentation	Missing one step for Content Script scheduling in Script Console
#001255	Online Documentation	Little error in Administrative page
#001232	Online Documentation	SAP extension: little changes in doc page
#001093	Online Documentation	Tag Guide of WebReport: there is an error
#001092	Online Documentation	Wrong method description in the API helper
#001089		

ID	Scope	Description
	Online Documentation	Beautiful Webforms views updater page: broken link and clarification
#001782	Module Suite	Smart View Task: Upload Area: Icon for shortcuts
#001766	Module Suite	Smart View Task: Upload Area: Icon for shortcuts
#001781	Module Suite	Error when using Custom Script widget in Smart Page
#001764	Module Suite	Error when using Custom Script widget in Smart Page
#001757	Module Suite	Error exporting remote webform with a template that has a Set with more than 1 row
#001737	Module Suite	Sidebar issues
#001717	Module Suite	Smart Page widget Container:Standard Nodetable not working starting from MS 3.4
#001087	Online Documentation	Classic UI page: broken links
#001085	Online Documentation	Broken link in admin page
#001084	Online Documentation	Doubt: two pages with instruction to how embed BWF in Smart UI
#001083	Online Documentation	Page Writing and executing scripts
#001079	Online Documentation	Content Script Extension for SAP: graphical issue
#001076	Online Documentation	Content Script editor: some problem and one error
#001075	Online Documentation	Page Content Server object: strange formatting and two footers
#001074	Online Documentation	Installing Smart Pages: is it possible review?
#001739	Module Suite	Missing inline documentation for new extension packages
#001773	Smart Pages	Custom Script Widget contains an error. Underscore library is not associated with the _ symbol.
#001289	Content Script	Tile News RSS Feed: when you add the widget and save, no code is added
#001728	Module Suite	Vulnerable JavaScript
#001763		Missing docman methods in the online help

ID	Scope	Description
	Online Documentation	
#001754	Module Suite	Document builder process issue
#001770	Module Suite	jQuery Interdependencies widget limits the number of form fields that can be selected for the Rule's Dependencies section
#001745	Module Suite	jQuery Interdependencies widget limits the number of form fields that can be selected for the Rule's Dependencies section
#001769	Module Suite	amgui doesn't display date according to language choose in settings
#001768	Module Suite	Issue on watermark
#001756	Module Suite	Issue on watermark
#001744	Module Suite	amgui doesn't display date according to language choose in settings
#001711	Smart Pages	SmartPages Custom Panel information
#001553	Smart Pages	The custom command in SmartUI does not work in custom search
#001694	Beautiful Webforms	It is not possible to set a dynamic default value to the Flatpickr widget (V4)
#001732	Module Suite	View and template do not change using Transfer Warehouse
#001731	Module Suite	View and template do not change using Transfer Warehouse
#001710	Beautiful Webforms	Checkboxes selection is ignored on Form reload
#001712	Beautiful Webforms	The 'SmartView Task Configuration' (V4) widget does not show the side panel when used outside of a workflow step
#001746	Beautiful Webforms	Handlebars Template Fix for Consistent Client and Server-Side Validation in Widgets
#001703	Content Script	"Make Favorite" error in "Functions" Menu
#001702	Content Script	"Make Favorite" error in "Functions" Menu
#001293	Beautiful Webforms	Creating a custom column having a content script as a data source does not terminate on PostgreSQL
#001706	Smart Pages	Smart View does not complete loading on xECM 23.3 when MS 3.5 is installed
#001402	Online Documentation	Module Suite Administration Tools - Select default IP address

ID	Scope	Description
#001405	Online Documentation	Typo error in Getting ready to upgrade Module Suite page
#001357	Online Documentation	Tooltip for buttons in CS Editor (very minor)
#001172	Online Documentation	Missing note about the path for the cs.log
#001077	Online Documentation	Minor typo errors or minor graphical issue
#001078	Online Documentation	Minor doubt on Form builder page
#001073	Online Documentation	Installation pages: minor issue
#001071	Online Documentation	Typo error in the left tree
#001663	Beautiful Webforms	Error Widget and Go Link Functionality Not Working in Forms with Multiple Tabs

Version 3.5.0 (Rome)- Release notes¶

Release Date	End of AMP(*)	End of Life
2023-08-02	2026-08-02	2027-08-02

(*) Active Maintenance Period

The present document contains information regarding product enhancements, fixed issues and known issues related to AnswerModules Modules Suite version 3.5.0.

This guide

The information presented in the on-line guide are mostly non-version specific. AnswerModules team does its best to ensure that, where necessary, is made clear that the information presented is only applicable to specific versions, however if you are looking for this version-specific documentation, you can find it here *(http://developer.answermodules.com/manuals/3.5.0)*

No Warranties and Limitation of Liability

Every effort has been made to ensure the accuracy of the features and techniques presented in this publication. However, AnswerModules accepts no responsibility and offer no warranty whether expressed or implied, for the accuracy of this publication.

Module Suite Compatibility Matrix¶

OpenText Content Server	MS 3.1.0	MS 3.2.0	MS 3.2.1	MS 3.3.0	MS 3.4.0	MS 3.5.0
Content Suite 16.2 EP6	X					
Content Suite 16.2 EP7	X					
Content Suite 20.2	X	X	X	X	X	
Content Suite 20.3	X	X	X	X	X	
Content Suite 20.4	X	X	X	X	X	
Content Suite 21.1	X	X	X	X	X	
Content Suite 21.2	X	X	X	X	X	
Content Suite 21.3	X	X	X	X	X	
Content Suite 21.4	X	X	X	X	X	
Content Suite 22.1		X	X	X	X	
Content Suite 22.2			X	X	X	
Content Suite 22.3				X	X	
Content Suite 22.4					X	
Content Suite 23.1					X(*)	
Content Suite 23.2						X
Content Suite 23.3						X

(*) Requires hotfix hotFix_ANS_340_010 to be installed

All Enhancements in version 3.5.0¶

ID	Scope	Description
#001699	Module Suite	It is now possible to control the logging level for classes annotated as @ContentScriptAPIService
#001291	Beautiful Webforms	Password field widget
#001664	Smart Pages	Added new windows10 icons
#001674	Module Suite	Release of New Widgets: Classifications, Toolbar, TreeView, ListView, and Grid
#001698	Smart Pages	Release of New Widgets: Toolbar and Grid
#001685	Module Suite	Introduction of CARL: AI agent for Enterprise Application Creation on XECM and Module Suite

ID	Scope	Description
#001678	Beautiful Webforms	Introduction of WCAG Compliant Widget Library Based on V4 Version
#001686	Module Suite	Introduction of "rmsec" Extension Package for Enhanced Record Management Security
#001697	Extension - PDF	New Feature - APIs for Text Extraction from PDF Files
#001696	Extension - Docx	Docx Extension Package Update: Revised Dependencies and Dropped Java 8 Compatibility
#001687	Module Suite	Introduction of "llm" Extension Package for LLM API Provider Integration
#001263	Content Script	Rolling of the cs.log file
#001695	Module Suite	Enhancements to Process Builder API - Introduction of "Step," "End," and Automatic Addition of Scripts and Forms
#001644	Content Script	The getLeader() method over a CSGroup object raises a NullPointerException if there isn't an user set as group leader
#001693	Module Suite	Adding Multiple Documents and Recipients to a CSEmail Object
#001692	Module Suite	New Feature: Internal API Now Supports File-Returning REST APIs
#001690	Module Suite	New "docman" APIs: getNodeData and getNodeDataAsJsonString for Node Information Retrieval
#001689	Module Suite	New API Introduced in DocmanService to Retrieve SubType Integer of Module Suite Objects
#001688	Module Suite	New APIs Introduced in CSDocument for Raw Content Extraction and Thumbnail Retrieval
#001680	Beautiful Webforms	Minor Fixes Required on SmartView Task View Template of Library V5
#001675	Module Suite	Deprecation of "View Smart Task Button" Widget and Introduction of "Smart View Task Configuration" Widget
#001681	Beautiful Webforms	Update on Code Generation from Form Builder: Snippet Storage Location Change
#001679	Beautiful Webforms	New Release: Enhanced SmartView Task View Template
#001677	Smart Pages	Update on V5 Tabs Widget: Specifying Active Tab and Executing Actions Upon Selection
#001624	Beautiful Webforms	Improved performances for smart-dropdown widgets (and its derivatives) by enabling client-side caching

Issues Resolved in version 3.5.0¶

ID	Scope	Description
#001554	Smart Pages	Issue if a Smart UI Custom Menu is added on multiple subtypes of the same parent
#001659	Content Script	Classic UI customizations for CSMenu don't work and return a generic Content Server error
#001670	Beautiful Webforms	Form.listformdata Method Fails and Generates Trace After Modifying Form Template
#001323	Content Script	Issue of the listFormData method of the forms service
#001285	Beautiful Webforms	Random validation error with the Phone widget
#001634	Extension - OAuth	OAuth service error messages incorrectly reference "AWS" Base Configuration profile instead of OAuth profiles
#001625	Module Suite	Generic HTTP 404 error opening Web Help from Content Script/ Smart Page Editor or from Form Builder
#001520	Content Script	Recman extension: removeOfficial method doesn't restore the original permission
#001591	Content Script	Module Suite Report: the section about Base Configuration is empty
#001656	Content Script	Insufficient permissions in Content Script Volume 'CSSmartView' folder causing unclear errors in cs.log
#001660	Content Script	Cache API connection hungs after a Content Server restart
#001646	Extension - Docx	The method replaceVariables of docx API removes white spaces replacing a placeholder
#001617	Content Script	Docx4j Marshaller: Issues parsing docx documents produced with the Sync extension
#001691	Module Suite	Fixed Issues with docman.getNodeRestV1JSon API and node.toJSONObject API
#001594	Content Script	Mail Service: Fetching issue for emails with attachments named containing "/" character
#001464	Beautiful Webforms	PWA: Exceptions are raised by the Vue devtools reference in the bwfv5.umd.min.js script.
#001682	Beautiful Webforms	Fix for Issue with FormBuilder Configuration Panel - Itemreference Configuration
#001580	Content Script	Custom logs are written to a wrong file

ID	Scope	Description
#001421	Content Script	Custom log appender: on rotation, a wrong data is used for the file name
#001652	Beautiful Webforms	User by Login Widget: Read-only mode allows value change using keyboard navigation
#001651	Beautiful Webforms	Graphical elements (like sort arrows and checkboxes) are not visualized for Datatable widget with OpenText template
#001616	Content Script	Page with Content Scripts list for Workflow: wrong label in remove alert popup
#001672	Module Suite	Issue with createDocument API in Recent Content Server Versions (20.X and above)
#001657	Content Script	When WebNodeActions script does not redirect to any page, a blank page may appear for some OTCS actions
#001599	Beautiful Webforms	Integer field in a Set are empty in Beautiful WebForm
#001619	Beautiful Webforms	Form Builder (Smart Editor) layout issue with V3 widget library (regression)
#001631	Extension - Rendition	Printing a BWF form to PDF using rend API contains visualization errors when user is not Admin
#001653	Extension - xECM	When referenced object does not exists, ECM initialization script fails avoiding startup of the Content Server service
#001622	Extension - Docx	Content Script fails with a generic 400 error setting properties in a MS Word document with method setProperties
#001642	Module Suite	Various Improvements and Bug Fixes for Application Builder
#001414	Online Documentation	Upgrading Module Suite: restore restart after each module update
#001639	Module Suite	The import tool does not notify that some widgets should be updated
#001627	Smart Pages	CSSmartView features: custom columns and custom commands are not shown with Module Suite running on Content Server 21.x
#001620	Beautiful Webforms	Issue with associating multiple forms having the same template with a workflow map
#001628	Smart Pages	Issue with the "Include SmartUI widget" widget: the data source script is never invoked
#001636	Module Suite	

ID	Scope	Description
		Scripting engine initializes without activation key and basic configuration
#001615	Content Script	The 'getGroupByName' method of the 'users' service does not work
#001626	Content Script	The getGroupByName method of the user service raises a generic OML Exception with Module Suite running on Content Server 21.x
#001623	Module Suite	The library import tool does not work properly on Windows in case OTCS has been installed under a path containing spaces.
#001609	Content Script	The getElementsByPath method raises a NullPointerException if called without root parameter for non administrator users
#001621	Content Script	The getGroupById method of the user service raises a ClassCastException if it's called using a user's identifier as a parameter
#001607	Content Script	DocBuilder: error generating PDF with Cyrillic alphabet characters

Version 3.4.0 (Rancate) - Release notes¶

Release Date	End of AMP(*)	End of Life
2023-02-02	2026-02-02	2027-02-02

(*) Active Maintenance Period

The present document contains information regarding product enhancements, fixed issues and known issues related to AnswerModules Modules Suite version 3.4.0.

This guide

The information presented in the on-line guide are mostly non-version specific. AnswerModules team does its best to ensure that, where necessary, is made clear that the information presented is only applicable to specific versions, however if you are looking for this version-specific documentation, you can find it here *(http://developer.answermodules.com/manuals/3.4.0)*

No Warranties and Limitation of Liability

Every effort has been made to ensure the accuracy of the features and techniques presented in this publication. However, AnswerModules accepts no responsibility and offer no warranty whether expressed or implied, for the accuracy of this publication.

Module Suite Compatibility Matrix¶

OpenText Content Server	MS 2.8.0	MS 2.9.0	MS 3.0.0	MS 3.1.0	MS 3.2.0	MS 3.2.1	MS 3.3.0	MS 3.4.0
Content Suite 16.2 EP6	X	X	X	X				
Content Suite 16.2 EP7	X	X	X	X				
Content Suite 20.2	X	X	X	X	X	X	X	X
Content Suite 20.3	X	X	X	X	X	X	X	X
Content Suite 20.4		X	X	X	X	X	X	X
Content Suite 21.1		X	X	X	X	X	X	X
Content Suite 21.2			X	X	X	X	X	X
Content Suite 21.3			X	X	X	X	X	X
Content Suite 21.4			X	X	X	X	X	X
Content Suite 22.1					X	X	X	X
Content Suite 22.2						X	X	X
Content Suite 22.3							X	X
Content Suite 22.4								X
Content Suite 23.1								X(*)

(*) Requires hotfix hotFix_ANS_340_010 to be installed

All Enhancements in version 3.4.0¶

ID	Scope	Description
#001588	Beautiful Webforms	Flatpickr: add internationalization support to Flatpickr to support for all languages
#001608	Extension - OAuth	Added the possibility of manipulating outgoing requests in the 'getAccessToken' API.
#001605	Script Console	The Script Console no longer requires a connection to import the OTCS configuration.
#001601	Beautiful Webforms	Date picker: add internationalization to support all languages
#001604	Beautiful Webforms	It is now possible to programmatically update the viewTemplate and the Smart Editor configuration of a view
#001583	Smart Pages	It is now possible to add custom panels between the properties of an object on Smart View

ID	Scope	Description
#001566	Module Suite	OData Service Improvements
#001564	Module Suite	Added Grid Widget in CSFormSnippets:V4:Sandbox and OData Service example

Issues Resolved in version 3.4.0¶

ID	Scope	Description
#001532	Online Documentation	Beautiful WebForm Update: import of libraries in volume is mandatory
#001584	Online Documentation	Script Console loadConfig: update the documentation page according with new import mode
#001587	Online Documentation	BWF updater: review the documentation according with the new tool
#001598	Online Documentation	Mobile WebForms: please remove the page
#001592	Content Script	Revoke of EDITPERMISSIONS, remove all the other permission
#001513	Beautiful Webforms	the change event is not being detected when using a Date Time Picker widget with an onChange widget
#001545	Beautiful Webforms	Multiple rows fields: it is not possible add/remote fields in the PreSubmit script
#001428	Module Suite	After upgrading MS to version 3.2.1, secret properties/ passwords in the basic configuration are lost
#001567	Beautiful Webforms	The title in the 'Widget Model' of the 'Box Container closed' Form Snippet is wrong
#001381	Script Console	If a OTCS User has a "!" in its password in a position different from the last character the Script Console login crashes
#001377	Beautiful Webforms	Switch Widget: actions configured under Data attributes are not triggered
#001586	Beautiful Webforms	The API 'listFormData' performs poorly when the submission mech is set to 'versions'.
#001569	Beautiful Webforms	Smart DropDown: issue setting default with multiple apostrophes characters
#001367	Beautiful Webforms	If in a workflow, hidden checkboxes on a Beautiful Web Form are reset.
#001385	Module Suite	Missing "References" menu entry for Content Script

ID	Scope	Description
#001321	Module Suite	Having many versions on a content script slows down the retrieval of objects
#001316	Smart Pages	Smart Menu: in the top bar, there is no way to show a string. Only icons is visible (regression from 3.0)
#001313	Beautiful Webforms	Signature Pad not working in 3.1 Version.
#001561	Smart Pages	CSSmartMenu: if there is a multiselection, there is no way to reset the counter of selected items
#001579	Beautiful Webforms	Datatables Search Builder: selecting Data field causes a JavaScript error and values are not passed to the backend
#001582	Module Suite	If a script used within a workflow has the character "_" in its name, its execution would result in an error
#001576	Content Script	After upgrade to 3.3, some workflow using Event Script stop to work with a generic error
#001578	Beautiful Webforms	TKL widget: popup is not closed after selecting a value
#001573	Beautiful Webforms	Sync Template Widget does not allow to specify an identifier and does not work when embedded in smart page
#001572	Beautiful Webforms	Grid widget does not generate the expected code in the OnLoad script
#001571	Smart Pages	Sync CSS is not properly applied when form is embeded in Smart Page
#001549	Smart Pages	Issue reinitialization Datatable in a form embedded in a SmartPage if there are multiple tiles in a perspective that embed forms
#001174	Smart Pages	Tree widget: Context aware option seems not working
#001376	Module Suite	GetNodeFast on an unexisting object raise an error
#001529	Beautiful Webforms	The Select From ViewParams widget (V4) does not work correctly if one of the values contains double quotes
#000912	Content Script	Issue retrieving listMembers of Esign groups
#001543	Content Script	Docx library generates XML files in pretty format when merging
#001557	Content Script	CSWS's otcs(Verb) (e.g. otcsGet) apis do not work on 22.3
#001565	Module Suite	FormBuilder and SmartPage Builder do not display widgets' help message in the configuration panel

ID	Scope	Description
#001559	Content Script	New CSScriptSnippets are not listed in the editor unless a search is performed.
#001556	Module Suite	Calling a CS RESTAPI where the script customizes the contentType of the response results in a blank page.
#001563	Module Suite	CSSmartMenu override not applied on 22.1

Version 3.3.0 (Montebello) - Release notes¶

Release Date	End of AMP(*)	End of Life
2022-11-01	2025-11-01	2026-11-01

(*) Active Maintenance Period

The present document contains information regarding product enhancements, fixed issues and known issues related to AnswerModules Modules Suite version 3.3.0.

This guide

The information presented in the on-line guide are mostly non-version specific. AnswerModules team does its best to ensure that, where necessary, is made clear that the information presented is only applicable to specific versions, however if you are looking for this version-specific documentation, you can find it here *(http://developer.answermodules.com/manuals/3.3.0)*

No Warranties and Limitation of Liability

Every effort has been made to ensure the accuracy of the features and techniques presented in this publication. However, AnswerModules accepts no responsibility and offer no warranty whether expressed or implied, for the accuracy of this publication.

Module Suite Compatibility Matrix¶

OpenText Content Server	MS 2.7.0	MS 2.8.0	MS 2.9.0	MS 3.0.0	MS 3.1.0	MS 3.2.0	MS 3.2.1	MS 3.3.0
Content Suite 16.2 EP6	X	X	X	X	X			
Content Suite 16.2 EP7	X	X	X	X	X			
Content Suite 20.2	X	X	X	X	X	X	X	X
Content Suite 20.3		X	X	X	X	X	X	X
Content Suite 20.4			X	X	X	X	X	X
Content Suite 21.1			X	X	X	X	X	X

OpenText Content Server	MS 2.7.0	MS 2.8.0	MS 2.9.0	MS 3.0.0	MS 3.1.0	MS 3.2.0	MS 3.2.1	MS 3.3.0
Content Suite 21.2			X	X	X	X	X	
Content Suite 21.3			X	X	X	X	X	
Content Suite 21.4			X	X	X	X	X	
Content Suite 22.1					X	X	X	
Content Suite 22.2						X	X	
Content Suite 22.3							X	

All Enhancements in version 3.3.0¶

ID	Scope	Description
#001527	Content Script	Content Script: option add version is not available
#001535	Content Script	Issue: The upgrade() method of the docman API returns always true
#001251	Online Documentation	Installation and upgrade page: highlight library import task
#001098	Module Suite	More robust form for license key
#001512	Beautiful Webforms	New features for the ADN widget
#001415	Content Script	Performance of the API isMemberOf: review and verify possible optimization
#001501	Module Suite	Content Script Result Tile is now using velocity macro for managing static dependencies

Issues Resolved in version 3.3.0¶

ID	Scope	Description
#001530	Beautiful Webforms	Select basic: if the values contains an & character, if an action trigger a reload, the value of that select is reset
#001424	Extension - SQL	The runSQL method of the sql API does not work if the cast of the input parameter is incorrect
#001375	Module Suite	AMXECM initialization error reported in thread logs
#001484	Content Script	Mail fetch: unable to retrieve attachments if there is an accent in the file name
#001493	Beautiful Webforms	Form Builder: in version 3.2.x is not showing widget of library V2, also if the library is present in the Volume
#001178		

ID	Scope	Description
	Extension - Rendition	Rend package has not been release for windows: on S3 there is only the linux one
#001384	Beautiful Webforms	Masking Script Error: $ is not recognized.
#001474	Beautiful Webforms	Smart View template: labels of fields are truncated if they are placed on top or bottom
#001472	Beautiful Webforms	jQuery Interdependencies: if it is set on a read only field, JavaScript error is raised and the page get stuck in loading
#001494	Beautiful Webforms	Custom Script Widget: function registerWidgetCallback non executed with SmartView template
#001406	Module Suite	Issue in docman.clonePermissions(..) API - "Public Access" right is restored on target object even if removed from source object
#001387	Beautiful Webforms	Issue in docman.clonePermissions(..) API - "Add major version" right is ignored
#001496	Content Script	Error setting a category attribute to nodes shared with Core Share
#001468	Beautiful Webforms	V4 Form template: am_grid.css and am_gridTable.css are missing
#001450	Beautiful Webforms	Layout widget generates a 404 error in Console/Network tab due am_gridTable.css missing
#001328	Beautiful Webforms	No login redirect if an custom action button is click and session is expired
#001500	Content Script	Content Server WebService getNode is not working and generate a trace with Module Suite 3.2.x
#001503	Content Script	LoadFormData fails if there are rows with the same
#001534	Smart Pages	Datatable widget not working in BWS widget on leading application (xECM)
#001531	Smart Pages	Forms that have a SubView are not rendered correctly when included in a Smart Page
#001524	Smart Pages	Regression with hotfix 020: if you open different Smart Pages, the user always see the content of first one
#001478	Beautiful Webforms	Handsontable widget: without setting "Grid height" property, dropdown fields are not usable
#001318	Script Console	Script console configuration can now be imported from the standard XML export of Module Suite Configuration
#001519	Module Suite	

ID	Scope	Description
		Issue with the path to anscontentscript temporary files in the Base Configuration page.
#001364	Beautiful Webforms	Modal container issue: after inserted into Form Builder, it is not possible remove, clone and configure
#001399	Module Suite	HTTP 401 error in a scheduled script that call a rest API
#001380	Module Suite	Conflict between AM patch and CGI patch
#001314	Beautiful Webforms	Smart DropDown: issue setting value in OnLoad if it contains not alphanumeric characters
#001429	Beautiful Webforms	On Content server 22.2 the 'User by login' widget does not work
#001430	Script Console	Script Console package is not present in 3.2.0 installer
#001510	Module Suite	Even if enabled, the Content Script Execution Auditing is not being tracked in the Audit table
#001518	Beautiful Webforms	Form Builder: not all the custom widgets are showed in the widget tree
#001517	Beautiful Webforms	Form Builder: if a custom widget is added in a BWF, when this form is edited, this widget is no more showed in editor
#001509	Smart Pages	Executing a search within the SmartUI OOB search feature for a Date Range belonging to a Category returns an error
#001525	Module Suite	When a workflow is transported to another environment references to scripts used in the workflow are lost
#001521	Module Suite	Module Suite is not logging on 22.3
#001470	Module Suite	Beautiful WebForms Studio-WebForm creation from PDF forms: Issue when selecting a pdf, javascript error stops the form creation
#001504	Beautiful Webforms	Beautiful WebForms Studio - Approval Application: An error is returned when override an existing application
#001505	Beautiful Webforms	Beautiful WebForms Studio: The currency widget in the BWF is not initialized correctly
#001523	Module Suite	Setting a list of users as a form step assignee in Process Builder generates a corrupted workflow map.
#001522	Module Suite	An error is raised when any sql code is executed (sql service) with parameters that are not strings
#001482	Beautiful Webforms	Smart Dropdown and OnChange action: if the dropdow has option "Use a single input", the onchange is not trigger after 2 items
#001307		Space Content context menu issue: not locked on the file

ID	Scope	Description
	Beautiful Webforms	
#001453	Content Script	New document created from a rendition or version: the version name is always set to csscript.txt
#001358	Content Script	When executed within a callback, renaming a Connected Workspace does not work.
#001471	Beautiful Webforms	Submit Button With Param doesn't send action parameters if there is an ADN dropdown field in the form
#001427	Beautiful Webforms	OnChange widget is not working with ADN Dropdown
#001417	Content Script	Impossible to modify Content Script step in a running workflow
#001476	Beautiful Webforms	Countries widget: using the V4 library, the flag icons are not shown
#001480	Beautiful Webforms	Datepicker: if current date is set in default widget with variable ${date.data} the field is empty on form load
#001479	Smart Pages	Action Button: if configured to perform the action Expand, Action Parameters are not populated
#001330	Smart Pages	Modal not opening in Smart Page
#001511	Module Suite	Content Script Volume Library Import Tool Might fail on Unix based systems
#001490	Smart Pages	Action button: The class of a button within a 'Button Container' is not set correctly
#001473	Content Script	Content Script Volume Import Tool page: error opening it if there isn't en_US in the Multilingual Metadata in Content Server
#001485	Beautiful Webforms	Smart DropDown DB Lookup: Callback feature is not working
#001454	Content Script	Synchronous callbacks NodeCopy and NodeMove are not interrupted throwing InterruptCallbackException
#001491	Module Suite	Content Script objects indexing does not work
#001419	Module Suite	Enabling Synchronous callbacks causes an error when creating new items
#001456	Content Script	Custom properties in base configuration: it is not possible to remove custom properties if they are marked as encrypted
#001455	Content Script	Custom properties in base configuration: it is possible to add the same property multiple times but with different values

ID	Scope	Description
#001351	Extension - Forms	i18n in Remote Web Form
#001434	Beautiful Webforms	Smart View Task template: if a pdf has an empty comment, it is showed with string "null" in comments tab
#001451	Content Script	In some cases, objects created via synchronous callback do not inherit permissions
#001486	Module Suite	Enterprise Connect stops to work after Module Suite 3.2.x upgrade
#001459	Module Suite	Issue with standard Content Server search in Smart UI using dates as filter, when MS is installed
#001487	Content Script	Velocity template error: Unable to create rendable form
#001422	Content Script	zip API: setPassword method is not working
#001378	Content Script	Error passing params to a Content Script through a WebReport step

Version 3.2.1 (Morcote) - Release notes¶

Release Date	End of AMP(*)	End of Life
2022-07-19	2025-07-19	2026-07-19

(*) Active Maintenance Period

The present document contains information regarding product enhancements, fixed issues and known issues related to AnswerModules Modules Suite version 3.2.1.

This guide

The information presented in the on-line guide are mostly non-version specific. AnswerModules team does its best to ensure that, where necessary, is made clear that the information presented is only applicable to specific versions, however if you are looking for this version-specific documentation, you can find it here *(http://developer.answermodules.com/manuals/3.2.0)*

No Warranties and Limitation of Liability

Every effort has been made to ensure the accuracy of the features and techniques presented in this publication. However, AnswerModules accepts no responsibility and offer no warranty whether expressed or implied, for the accuracy of this publication.

Module Suite Compatibility Matrix¶

OpenText Content Server	MS 2.6.0	MS 2.7.0	MS 2.8.0	MS 2.9.0	MS 3.0.0	MS 3.1.0	MS 3.2.0	MS 3.2.1
Content Suite 16.2 EP6		X	X	X	X	X		
Content Suite 16.2 EP7	X	X	X	X	X	X		
Content Suite 20.2		X	X	X	X	X	X	X
Content Suite 20.3			X	X	X	X	X	X
Content Suite 20.4				X	X	X	X	X
Content Suite 21.1				X	X	X	X	X
Content Suite 21.2					X	X	X	X
Content Suite 21.3					X	X	X	X
Content Suite 21.4					X	X	X	X
Content Suite 22.1							X	X
Content Suite 22.2								X

All Enhancements in version 3.2.1¶

ID	Scope	Description
#001448	Extension - xECM	New API to get the Workspace directly from a Business Workspace node
#001443	Content Script	Introduced Module Suite health check page among administrator settings
#001439	Content Script	Introduced verification of scripting engine activation status at startup. Initialization scripts are not executed on an inactive
#001438	Module Suite	mproved initialization for the Module Suite template engine. Initialization of singleton objects has been synchronized.
#001436	Content Script	New API for accessing the volume of "Document templates"
#001394	Module Suite	Page Manage Callbacks: error raised if search is performed without select an object

Issues Resolved in version 3.2.1¶

ID	Scope	Description
#001389	Module Suite	Unable to set default value for Smart DropDown DB Lookup if value contains not alphanumeric chars

ID	Scope	Description
#001449	Smart Pages	SmartPages are not cached correctly by the templating engine
#001423	Beautiful Webforms	Form Template View "OpenText" Do not show header icon
#001447	Extension - xECM	You can now create a Business workspace in any space if the Business workspace type is configured this way.
#001446	Extension - PDF	Fixed the API for applying a watermark to a PDF (rotation is now in degrees)
#001444	Module Suite	The volume import tool does not detect differences between imported and incoming objects if they have the same version.
#001442	Content Script	Since 22.2, the presence of a % in a runSql* API parameter generates an error
#001440	Content Script	Resolved problems with layered configuration not honored by standard administration settings import.
#001437	Content Script	Improved caching policies for APIs that grant direct access to "Volume" type nodes, e.g., category volume.
#001435	Beautiful Webforms	Classification (199) and Classification Tree (196) type objects are displayed with an incorrect icon in the NodeTable widget.
#001349	Extension - Forms	Remote Form content: not drop area with IE
#001395	Module Suite	Page managelog.cs: wrong label and script name truncated
#001397	Module Suite	Manage Callbacks search page: it is not possible select business workspace
#001392	Module Suite	Unable to use the page Manage Callbacks search form if Enable check Next URL is enabled
#001382	Module Suite	If the file name provided in the creation dialog ends with ".cs," OTCS may generate an error.
#001298	Smart Pages	Possible issue with flyout option in Smart UI Menu
#001393	Module Suite	Unable to use the page Manage Callbacks for Oracle DB
#001416	Smart Pages	Custom Columns in SmartView: performance issue when applied on Virtual Folder
#001412	Module Suite	Form Builder lists widgets that cannot be displayed, if widgets from the current library have not been imported into CSVolume
#001410	Module Suite	The "Missing Widget" placeholder is not rendered correctly in Form Builder

ID	Scope	Description
#001409	Module Suite	Form Builder does not correctly create the default view if the current library widgets have not been imported into the CSVolume

Version 3.2.0 (Locarno) - Release notes¶

Release Date	End of AMP(*)	End of Life
2022-04-15	2025-04-15	2026-04-15

(*) Active Maintenance Period

The present document contains information regarding product enhancements, fixed issues and known issues related to AnswerModules Modules Suite version 3.2.0.

This guide

The information presented in the on-line guide are mostly non-version specific. AnswerModules team does its best to ensure that, where necessary, is made clear that the information presented is only applicable to specific versions, however if you are looking for this version-specific documentation, you can find it here *(http://developer.answermodules.com/manuals/3.2.0)*

No Warranties and Limitation of Liability

Every effort has been made to ensure the accuracy of the features and techniques presented in this publication. However, AnswerModules accepts no responsibility and offer no warranty whether expressed or implied, for the accuracy of this publication.

Module Suite Compatibility Matrix¶

OpenText Content Server	MS 2.5.0	MS 2.6.0	MS 2.7.0	MS 2.8.0	MS 2.9.0	MS 3.0.0	MS 3.1.0	MS 3.2.0
Content Suite 16.2 EP6	X		X	X	X	X	X	
Content Suite 16.2 EP7		X	X	X	X	X	X	
Content Suite 20.2			X	X	X	X	X	X
Content Suite 20.3				X	X	X	X	X
Content Suite 20.4					X	X	X	X
Content Suite 21.1					X	X	X	X
Content Suite 21.2						X	X	X
Content Suite 21.3						X	X	X
Content Suite 21.4						X	X	X

OpenText Content Server	MS 2.5.0	MS 2.6.0	MS 2.7.0	MS 2.8.0	MS 2.9.0	MS 3.0.0	MS 3.1.0	MS 3.2.0
Content Suite 22.1								X

Major Changes in version 3.2.0¶

Content Script Volume management¶

Prior to Module Suite version 3.2, all Content Script Volume resources had to be necessarily imported in the Volume, with no exceptions. Starting with version 3.2, Module Suite is capable of using certain resources (CSFormSnippets, CSScriptSnippets, CSPageSnippets) directly from the Module installation folders on the filesystem, without the strict need to "materialize" them in the Content Script Volume. This approach allows to avoid the overhead of importing certain resources if the administrator does not plan to customize them, but it optionally allows to "materialize" them in the Volume if needed.

This new approach allows to significantly reduce the effort required in validating the content of the Content Script Volume and solving conflicts in case of updates, since if the resources have not been materialized, the update will be transparent for the users (the library in the new Module version will replace the old one).

As a result of this new approach, the CSVolume administration tools have been reorganized and updated.

See the Content Script Volume Import Tool guide for additional details.

Issues Resolved in version 3.2.0¶

ID	Scope	Description
001314	Beautiful Webforms	Issue on Smart DropDown - special characters in option values
001359	Beautiful Webforms	When removing a Box Container widget from the form builder, the next widget is deleted too
001362	Beautiful Webforms	Cloned widgets are removed from the view upon saving
001339	Beautiful Webforms	Error in PDF viewer rendition if file name contains special characters
001338	Beautiful Webforms	Unable to configure Smart DropDown DB Lookup - field values cannot be selected
001337	Beautiful Webforms	Datatable: inline menu buttons are not visible

ID	Scope	Description
001335	Beautiful Webforms	Iteration container widget is removed from view upon saving and reopening the Form Builder
001279	Beautiful Webforms	Form Builder Toolbar gets cut after moving widget at the bottom of the grid
001315	Beautiful Webforms	Mapping Script widget not working (BWF library V4)
001352	Beautiful Webforms	The 'Wysiwyg Editor' widget (BWF library V4) is not displayed correctly in ReadOnly mode
001309	Beautiful Webforms	After update to MS 3.1, Form Builder drops closing element of Container widgets
001312	Beautiful Webforms	After update to MS 3.1, unable to edit BWF view built with BWF library V3
001310	Content Script	The 'isChain' parameter is ignored when programmatically scheduling the execution of Content Scripts
001361	Content Script	A node's nickname value is not loaded correctly if the node information is loaded using a lazy-access API
001305	Content Script	Traces are generated when using csws API to call webservices
001296	Extension - Docx	Obsolete log4j file subject to vulnerability is included in 'docx' service library
001297	Content Script	Content Script 'template' service fails to initialize after upgrade to Module Suite 3.1
001308	Extension - Docx	createSpreadsheet() API method of the xlsx service throws exception
001326	Extension - Forms	Beautiful WebForms Studio: wizard for Remote Form export stops at 'Working area' step
001332	Module Suite	CORS related issues for pages and forms when embedded in leading application
001129	Smart Pages	Pagination issues in Node Table widget - navigation reset to page 1 whenperforming back action
001327	Smart Pages	Fragment does not work
001336	Script Console	Obsolete log4j file subject to vulnerability is included in library

Version 3.1.0 (Ascona) - Release notes¶

Release Date	End of AMP(*)	End of Life
2022-01-15	2025-01-15	2026-01-15

(*) Active Maintenance Period

The present document contains information regarding product enhancements, fixed issues and known issues related to AnswerModules Modules Suite version 3.0.0.

This guide

The information presented in the on-line guide are mostly non-version specific. AnswerModules team does its best to ensure that, where necessary, is made clear that the information presented is only applicable to specific versions, however if you are looking for this version-specific documentation, you can find it here *(http://developer.answermodules.com/manuals/3.1.0)*

No Warranties and Limitation of Liability

Every effort has been made to ensure the accuracy of the features and techniques presented in this publication. However, AnswerModules accepts no responsibility and offer no warranty whether expressed or implied, for the accuracy of this publication.

Module Suite Compatibility Matrix¶

OpenText Content Server	MS 3.1.0
Content Suite 16.2 EP6	X
Content Suite 16.2 EP7	X
Content Suite 20.2	X
Content Suite 20.3	X
Content Suite 20.4	X
Content Suite 21.1	X
Content Suite 21.2	X
Content Suite 21.3	X
Content Suite 21.4	X

Major Changes in version 3.1.0¶

All Enhancements in version 3.1.0¶

ID	Scope	Description
#001140	Beautiful Webforms	Disable ADN on page reload
#001191	Content Script	When saving a script from the Content Script Editor also the Content Suite Static Variables should be saved.
#001170	Content Script	Library update procedure: folders are often skipped
#000961	Smart Pages	Missing methods to update Physical Object
#000960	Smart Pages	Ordering in custom commands
#001256	Beautiful Webforms	"PDF Viewer: when a document is downloaded
#001179	Smart Pages	Smart UI Accessibility Issues for people with disabilities
#001155	Beautiful Webforms	Smart View Task view template now supports adding documents and shortcuts to WF's attachments from OTCS.
#001054	Content Script	Rename a folder with internationalization activated
#001167	Content Script	New API service for updating the table of a form template
#000947	Module Suite	Re-import of a Content Script is not supported
#001183	Module Suite	Activation Key information is no longer persisted on INI file

Issues Resolved in version 3.1.0¶

ID	Scope	Description
#001160	Online Documentation	Possible confusion in the release note page
#001224	Online Documentation	Content Script Node Table Tile: the code used as example is wrong
#001113	Online Documentation	Add to documentation property to fix the address
#001169	Online Documentation	Installation on multiple server
#001242	Online Documentation	Remove Web Form: copy al support folders

ID	Scope	Description
#001253	Online Documentation	Crete a page or a specific paragraph for custom template and snippets
#001201	Content Script	Sharing issue with Coreshare
#001270	Content Script	Issue of the getClassificationNode() method of the recman service
#001214	Beautiful Webforms	Footer section missing in Modal Container widget
#001261	Beautiful Webforms	Image widget issue
#001269	Beautiful Webforms	"Flatpickr in Smart View: if it is present in one page
#001286	Beautiful Webforms	Base configuration show password in additional properties
#001283	Beautiful Webforms	V5: Form is not rendered when there's a Text Popup Form Template field in the model
#001284	Beautiful Webforms	Fix validators for V5 library
#001276	Content Script	Helper: the documentation for the docman API is missing
#001275	Beautiful Webforms	The View Smart Task Button widget is not visible
#001278	Beautiful Webforms	V5 library: when using Smart View Task template the submit of the form retruns an error
#001287	Beautiful Webforms	Add property to manage TLS version for mail service
#001285	Beautiful Webforms	Random validation error with the Phone widget
#001249	Beautiful Webforms	Graphic issue on the configuration of the 'Buttons Group' widget
#001246	Beautiful Webforms	Minor usability issue: alignment difference of a label between editor and form
#001250	Beautiful Webforms	Graphic issue on the configuration of the 'Table' widget
#001248	Smart Pages	It is not possible to set the visibility of the 'Box Container' widget
#001244	Extension - Forms	

ID	Scope	Description
		Missing a default out of the box template for Remote Web Form
#001240	Extension - Forms	Issue on method updateTable
#001238	Beautiful Webforms	JavaScript Error adding debug box widget
#001229	Beautiful Webforms	Usability issue:18n checkbox available where it should not
#001225	Smart Pages	"For Content Script under CSSmartView:Commands folder
#001220	Beautiful Webforms	'Bold Label' checkbox missing in the 'Space content' widget
#001219	Beautiful Webforms	Missing icons in button widgets
#001210	Module Suite	CSSmartView Column: adding back compatibility with 2.9
#001196	Beautiful Webforms	V3 Buttons Group Visibility Rules
#001204	Content Script	Menu lazy doesn't work in the 'Content Script Nodes Table' of the perspective
#001193	Beautiful Webforms	Currency field: strange behavior if comma is set as decimal separator
#001176	Beautiful Webforms	Online editor has a wrong link
#001061	Content Script	Catch Exceptions thrown from different script
#001110	Beautiful Webforms	BWF endpoint is unable to deserialize form object if form.viewParams contains classes that have been defined within a Script
#001138	Extension - xECM	Helper: the documentation for the XECM API is missing
#001064	Beautiful Webforms	Add internationalization support to Datepicker
#001066	Beautiful Webforms	Missing file size in Space Content after upload
#001118	Smart Pages	"CSSMARTMENU : custom menu items missing in search results view with Tabular search view"""
#00959	Content Script	CSTaskImpl.assignedTo doesn't work
#001226	Content Script	Random error Unable to find resource '/AMST-1027490201'

ID	Scope	Description
#001230	Beautiful Webforms	It is no longer possible to add a field to a set using the FormBuilder
#001158	Content Script	Little change in editor after upgrade
#001101	Smart Pages	"Datatable widget doesn't support client side actions (like pagination
#001055	Content Script	Minor error with online helper
#001050	Beautiful Webforms	Issue mapping name of column on Table widget
#001049	Beautiful Webforms	Issue Users in Group widget
#001048	Beautiful Webforms	Issue on Dropdown and Service on Handsontable widget
#001047	Module Suite	In the Task object it's possible to create a Module Suite Template
#001036	Beautiful Webforms	Two Progress Bar form snippets
#001035	Content Script	Incorrect Widgets CSSynchEvent
#001032	Beautiful Webforms	Issue Clear button on Smart DropDown widget in read only mode
#001026	Beautiful Webforms	Some incorrect SmartUI Widgets (v3)
#00941	Beautiful Webforms	Smart DropDown and select has a very little style glitch
#001012	Beautiful Webforms	"In Beautiful WebForm
#001011	Smart Pages	"In Smart Pages
#00966	Beautiful Webforms	Adding a row on Smart DropDown using the template SmartView on Firefox doesn't work
#00378	Extension - Docx	"In certain cases
#001175	Content Script	CSWS and pool widget not working with 21.3
#001273	Beautiful Webforms	V5 library: an easy from with only Space Content is not rendered due JS error
#001185	Beautiful Webforms	Issues editing views that have been transported

ID	Scope	Description
#001129	Smart Pages	Page in Smart View with node table always back on page 1 in case of multiple page
#001265	Beautiful Webforms	"Scheduling option reset to default from the ""Specific"" context menu"
#001215	Smart Pages	CSSmartView:Columns not displayed on Results page
#001243	Beautiful Webforms	Usability issue in Select Basic (see screenshot)
#001222	Beautiful Webforms	Label issue of the Radio Basic widget
#001203	Beautiful Webforms	Comments missing in the SmartView Task template
#001281	Extension - sFTP	Private key is visible in the log
#001266	Beautiful Webforms	Default value for Flatpk and date picker is not working
#001264	Beautiful Webforms	Usability issue in Panel Layout
#001259	Beautiful Webforms	"Panel Layout: problem in the form builder if ""is collabsible"" is checked"
#001231	Beautiful Webforms	Forms having revision mech specified are not properly persisted when retrieved using forms.getFormInfo
#001211	Beautiful Webforms	Graphic issue of the loading indicator of the Space Content widget
#001180	Content Script	duplicate row creation when initiating a workflow form content script
#001272	Module Suite	Issue on the perspectives that include a Smart Page
#001280	Module Suite	Critical security vulnerability related to log4j CVE-2021-44228 / CVE-2021-45046
#001044	Module Suite	Regression 1013: Base configuration custom props are not initialized
#001233	Beautiful Webforms	Add internationalization support to Flatpickr
#001202	Beautiful Webforms	Edit button missing in the attachments of the SmartView Task template
#001217	Smart Pages	"scope : ""single"" in a Smart Menu is ignored"

ID	Scope	Description
#001223	Smart Pages	Tile Content Script Nodes Table: wrong code inserted by snipped
#001227	Beautiful Webforms	User by login: translation is not working
#001228	Beautiful Webforms	"SmartDrop down: if no result in filter
#001236	Smart Pages	Content Script Result: css issue
#001199	Beautiful Webforms	i18n in select basic is not working
#001213	Extension - Forms	Error in process of export of a Remote Form
#001209	Module Suite	"Issue creating pdf of a form generated by ""Beautiful WebForm Studio"""
#001151	Smart Pages	"Datatable: if it is enabled the drop area
#001173	Script Console	Error 500 adding a new script with same name
#001189	Smart Pages	Smart Page actions are cumulated when using navigation between Smart View Perspectives
#00680	Content Script	Accessing rendition content on CSVersion result in wrong content
#001271	Beautiful Webforms	Communication between smart pages
#001200	Extension - Docx	Issue with html field into docx document
#001234	Content Script	Under particular circumnstances a script executed by DA might lead to a system freeze till the operation is completed
#001188	Content Script	"Issue on ""Always run impersonating"" user"
#001190	Smart Pages	Panel layout Widget in SmartPages Page Builder is missing configuration text boxes
#001182	Beautiful Webforms	Issue when importing the template view through the Transport Warehouse
#001192	Module Suite	When filtering widgets or snippets in IDE if user clciks on Submit/Enter the page refreshes and shows Enterprise Workspce
#001184	Beautiful Webforms	"When a version of a view is deleted
#001241		Custom HTML form template not visible in the Form Builder

ID	Scope	Description
	Beautiful Webforms	
#001187	Content Script	When typing in the Content Script Static Variable Tabs window flickers
#001186	Content Script	Minor issue of Run SQL and Run SQLFast widgets
#001177	Smart Pages	Smart UI widget title
#00884	Beautiful Webforms	Issue Wysing Editor the copied image is duplicated
#001136	Beautiful Webforms	Space content spin load: graphical issue in Smart View and Smart view task
#001168	Beautiful Webforms	Change behavior in the hidden text field
#001127	Beautiful Webforms	Currency field doesn't trigger OnChange
#001027	Script Console	Little error in installer for 2.8.0
#001161	Beautiful Webforms	Change in multi-field behavior: clear is not working
#001162	Content Script	Smart Menu doesn't work after upgrade to 3.0.0
#00838	Content Script	Workflow Suspended leads to a blank Content Script Step
#001159	Smart Pages	Tile Content Script node table result is not working in 3.0.0
#001156	Beautiful Webforms	"Space content: uploading a file
#001150	Beautiful Webforms	On Event Validation widget: it is not possible select the field
#001153	Beautiful Webforms	Include SmartUI Widget Widget fails because region's 'el' is not already loaded in page
#001154	Smart Pages	Include SmartUI Widget fails on 16.2.8
#001152	Beautiful Webforms	ADN ID widget is missing Content Script Snippet
#001141	Extension - ZIP	"Regression on ZipContext
#001145	Smart Pages	"SmartPage: if a template is selected for the smart page
#001149	Beautiful Webforms	"Date fields
#001146		Smart DropDown DB Lookup is not working in 3.0.0

ID	Scope	Description
	Beautiful Webforms	
#001144	Smart Pages	Error in contentScript script: it is failing the version check for 16.2.8
#001147	Beautiful Webforms	No page reload/action triggered if there is a subview
#001142	Beautiful Webforms	Show-if conditions not properly evaluated within Sets on V5
#001094	Beautiful Webforms	Default in Modal Container
#001102	Smart Pages	Issue title of the confirmation dialog of DataTable widget
#001143	Beautiful Webforms	Downloading the Excel Template from BWF Form Studio results in a corrupted file
#001148	Beautiful Webforms	"Adding a row to a set in a form where data was already submitted

Version 3.0.0 (Generoso) - Release notes¶

Release Date	End of AMP(*)	End of Life
2021-06-30	2024-06-30	2025-06-39

(*) Active Maintenance Period

The present document contains information regarding product enhancements, fixed issues and known issues related to AnswerModules Modules Suite version 3.0.0.

This guide

The information presented in the on-line guide are mostly non-version specific. AnswerModules team does its best to ensure that, where necessary, is made clear that the information presented is only applicable to specific versions, however if you are looking for this version-specific documentation, you can find it here *(http://developer.answermodules.com/manuals/3.0.0)*

Script Console Installer

The Script Console installer has been temporarily removed from the Module Suite master installer. It will be reinstated in the next minor release.

No Warranties and Limitation of Liability

Every effort has been made to ensure the accuracy of the features and techniques presented in this publication. However, AnswerModules accepts no responsibility and offer no warranty whether expressed or implied, for the accuracy of this publication.

Module Suite Compatibility Matrix¶

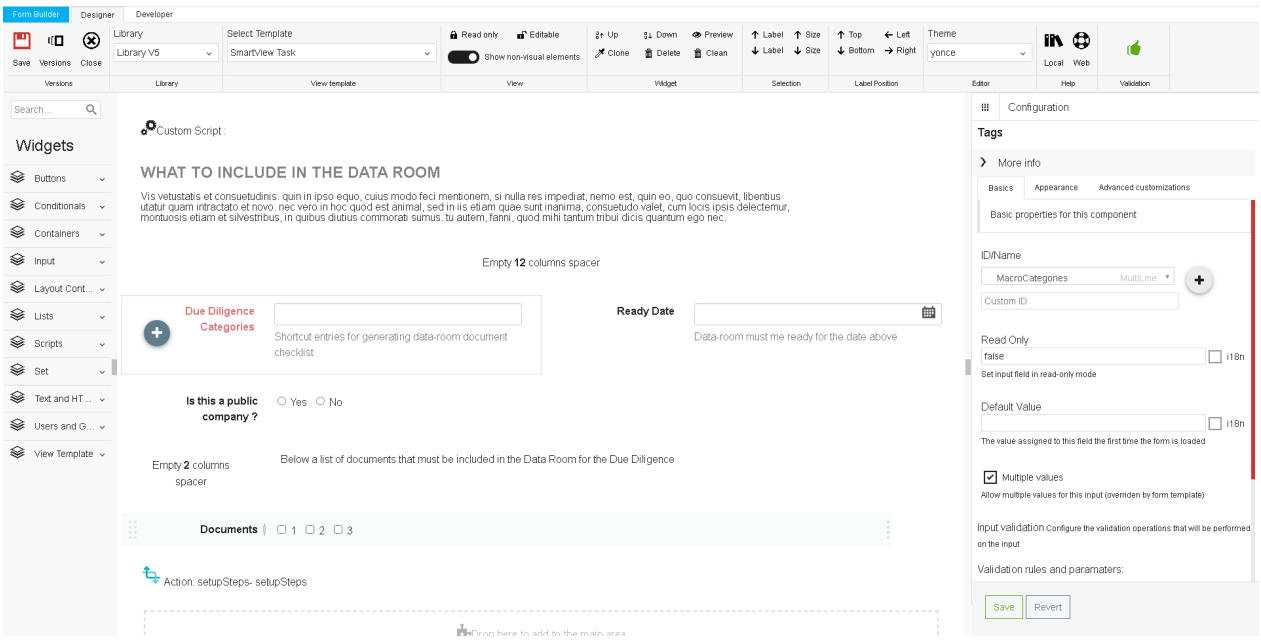
OpenText Content Server	MS 3.0.0
Content Suite 16.2 EP6	X
Content Suite 16.2 EP7	X
Content Suite 20.2	X
Content Suite 20.3	X
Content Suite 20.4	X
Content Suite 21.1	X
Content Suite 21.2	X
Content Suite 21.3	X
Content Suite 21.4	X

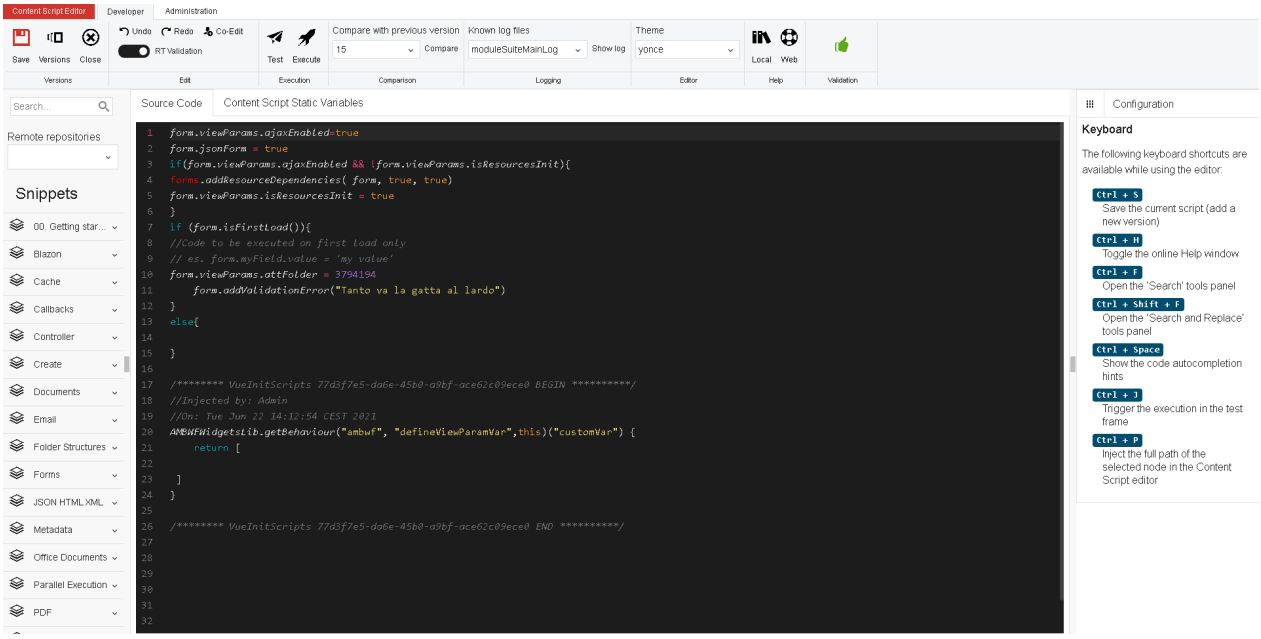
Major Changes in version 3.0.0¶

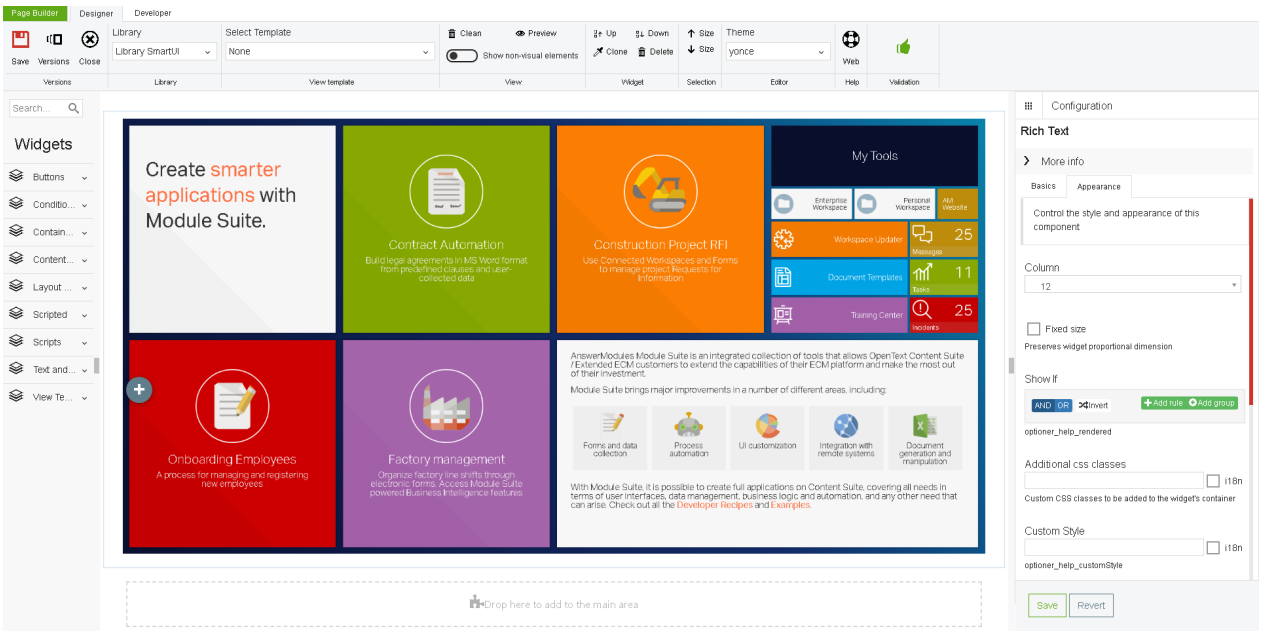
IDEs¶

All the Module Suite's IDEs have been deeply revised. Among the new functionalities introduced: filtering for snippets and widgets, editor theme selector, log level rapid switch for Content Script Editor, remote repositories for Content Script snippets, Content Script Co-edit (Beta)

Beautiful WebFormsContent Script EditorPage Builder

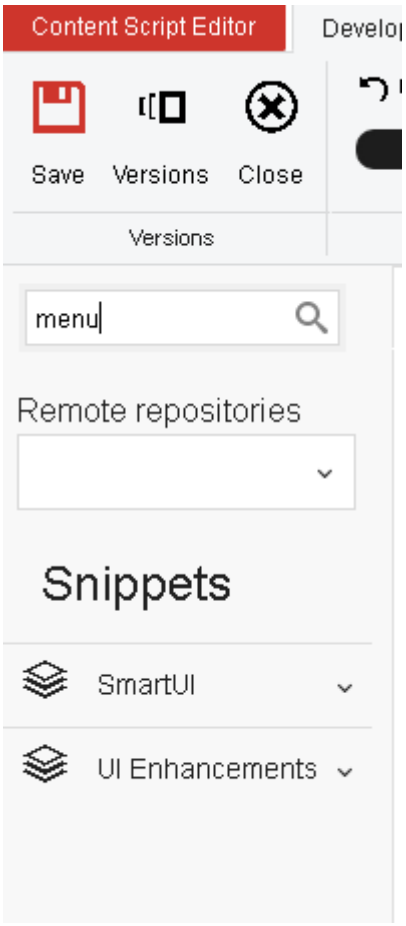






Filtering¶

A new filtering feature has been added to all IDEs to make it easier to select the appropriate widget or snippet in large libraries.



Remote snippets repositories¶

You can now retrieve Content Script snippets from remote repositories. This allows you to maintain an enterprise KB related to Content Script (in the form of a local Snippets repository or leverage Snippet repositories offered by third-party vendors. To register a new repository you need to add a custom option in Base Configuration having the form: `amcs.msrepo[n].url=Label|repoUrl` where `n` is a number between 0 and 10.

```
amcs.msrepo0.url=Sandbox|https://developer.answermodules.com/resources/repos/sandbox
```

Concurrent Script Editing¶

Module Suite 3.0 features an experimental functionality that allows several developers to simultaneously collaborate on the editing of the same script. The functionality leverages WebRTC to establish a peer-to-peer direct connection among developers. The developer's browser will connect to the specified signaling server to find other peers. A password can be specified to encrypt all communication on the signaling server even if no sensitive information (WebRTC connection information, shared data) is shared over the signaling server.

Content Script¶

Updated of all major dependencies to their latest releases. New APIs for creating and manipulating OTEmail objects and OT Pulse comments. Improvements to performances related to the retrieval of information from the database.

Administration¶

New performances tuning options available in the Module Suite base configuration.

Beautiful WebForms¶

New V5 library¶

Module Suite 3.0 introduces a new widget libary based on reactive components (Vue.js *(https://vuejs.org/)*). With this library, the already powerful engine, used to perform server-side rendering of forms' views is complemented by a reactive framework operating directly in the user's browser. When a form's view is composed using this library, the data model that is normally used in server-side rendering (form) is also serialized into a JSON object on the user's browser. This client side "model" feeds a reactive application developed with Vue.js *(https://vuejs.org/)*. Thanks to this new approach we open up the possibility of performing numerous manipulations of the data model directly on the user's client.(i.e. it is no longer necessary to

perform a client-server round-trip to manipulate the data-model), which do not longer require to update (totally or partially) the page containing the view. To support and facilitate the manipulation of the data model on the user's browser, the concept of `action`, already in use for server-side manipulation of the data model, has been extended and revised. When an action is now triggered the frameworks looks for its implementation first in a client-side registry, and only if it is not found proceeds invoking the server-side business logic (CLEH). The implementation of a client-side action is pretty simple and can leverage a dedicated javascript API, whose main methods are:

```
form //represents the form object (as in CLEH scripts)
form.validate() //Triggers form validation
form.getFieldReference(index, fieldName) //Access the input widget associate to a specific form's fi
                                         //fieldName is the field's path in the form (e.g. MySet:MyF
                                         //index represent the set row
form.viewParams // The viewParams variable as in CLEH scripts
                // e.g. form.viewParam.vmVar
form.submitForm(withValidation) //Submits the form eventually triggering the form's validation first
form.getFieldValues(fieldName) //Retrives the list of values for the given form's field
form.getViewParamsValue(viewParamName) //Retrieves the value associated to the given viewParams's va
                                       //The main difference between form.viewParams.myVar and form.
                                       //is that if myVar contains an object having the following st
                                       // {ajax:{url:"https://some.service.com/endpoint", data:[]}}
                                       //the API form.getViewParamsValue('myVar') automatically fetc
                                       //remote service and caches the result in the objects 'data'
form.setViewParamsValue(variable, value) //Set the value of a viewParams variable
form.setFieldValues(fieldName, values) //Set the values for the given field
form.setFieldReadOnly(fieldName, values) //Set the field as read-only or editable
form.addField(fieldName, index) //Adds an instance to the specified field
form.removeField(fieldName, index) //Remove an instance to the specified field
form.addConstraint(fieldName,contraint, configuration) //Adds the specified validation constraint to
form.removeConstraint(fieldName, contraint) //Removes the specified validation constraint to the giv
```

CLEH scripts

If an action is triggered but it can not be found among the registered client side actions, we assume it is a server side action and the CLEH script is executed allowing server side manipulation of the data-model

New widgets for library V4¶

Added new widgets in library V4

Smart Pages¶

Commands definition cache¶

It is now possible to cache (using the distributed memcache) the result of the execution of the scripts stored under "CSSmartView:Commands" used to load the definitions of the additional commands you want to be available in Smart View pages. The scripts outcome is cached on a per-user basis. To enable the caching set to true the "amcs.amsui.volumeCache" parameter in Base Configuration. To programmatically clean the cache use the amsui.clearCache() API.

Actions definition cache¶

It is now possible to cache (using the distributed memcache) the list of scripts under "CSSmartView:Actions" used for lazy loading additional commands in the Smart View pages. The scripts list is cached on a per-user basis. To enable the caching set to true the "amcs.amsui.volumeCache" parameter in Base Configuration. To programmatically clean the cache use the amsui.clearCache() API.

Overrides optimization¶

The internal mechanisms related to how the customizations are applied to the menus and the columns of the browsing pages of the Smart View interface have been deeply revised. The content of the Overrides folder is now used to compute an Override Map (OM), specific to your repository, having the following structure:

```
OM = [
    "globals": [            (1)
        540588
    ],
    "type": [               (2)
        "144": [            (3)
            548066
        ]
    ],
    "tenants": [            (4)
        "497147": [         (5)
            "globals": [    (6)
                548169
            ],
            "type": [       (7)
                "144": [    (8)
                    496932
                ]
            ],
            "ids": [        (9)
                "496931": [ (10)
                    545972
                ]
            ]
        ]
    ]
]
```

where:

- (1) identifies a list of scripts to be always executed
- (2) a list of scripts to be executed only if the current space has at least one node having of the identified type (3)
- (4) scripts to be considered only if the current space is descendant of the specified tenant (5) (a space identified by its DataID)
- (5) is a "tenant" configuration
- (6) identifies a list of scripts that must always be executed if the current space is descendant of the specified tenant (5)

- (7) a list of scripts to be executed only if the current space has at least one node having of the identified type (8) and is descendant of the specified tenant (5)
- (9) a list of scripts to be executed only if the current space has at least one node having of the identified id (10) and is descendant of the specified tenant (5)
- scripts in the OM are executed in the following order (1), (2), (6), (7), (10).

Given the above example and imagining that all the scripts in (3) (8) and (10) return the list ["comm_one","comm_two"], the resulting AOM will contain:

```
(3) AOM = [
                ...
                "S144":[commands:["comm_one","comm_two"]],
                ...
            ]
    (8) AOM = [
                ...
                "S144":[commands:["comm_one","comm_two"]],
                ...
            ]
    (10) AOM = [
                ...
                "D496931":[commands:["comm_one","comm_two"]],
                ...
            ]
- scripts in (1), (6), (10) MUST return a Map having entries of the form:
    "SXXXX":[
        commands:["comm_one", "comm_two",...],
        columns: [ //Optional
                    col_name:"col value", //value can be HTML
                    ...
                    ]
    ]
    where XXXX is a valid SubType
    or
    "DYYYY":[
        commands:["comm_one", "comm_two",...],
        columns: [  //Optional
                    col_name:"col value", //value can be HTML
                    ...
                    ]
    ]
```

where YYYY is a valid node's ID.

OM is to be considered a "static" information in productive environments and as such, to guarantee optimal performances, the framework should be allowed to cached it by setting to "true" the " amcs.amsui.volumeCache" parameter int the base configuration.

When a user changes the current space, the OM is evaluated by the framework against the users' permissions and the actual override map (AOM) associated to the space is determined. AOM is determined by executing the relevant scripts in OM in the order described above. The AOM has the following form:

```
AOM = [
    "S144":[                                                                    (1)
            commands:["comm_one", "comm_two",...], //list of commands' command_key  (2)
            columns: [                                                          (3)
```

```
                        col_name:"col value", //value can be HTML
                        ...
                    ]
            ],
    "D1234":[                                                                (4)
            commands:["comm_one", "comm_two",...], //list of commands' command_key
            columns: [
                        col_name:"col value", //value can be HTML
                        ...
                    ]
            ]
    ...
]
```

where: (1) represents commands and columns to be associated to all the nodes having the identified subtype, (3) can be omitted, (4) represents commands and columns to be associated a specific node (identified by its id), (4) takes precedence over (1).

How OM is created ?¶

In order to determine the OM, the content of the "Overrides" folder is evaluated following the logic below:

```
[
    "globals":[             (1)
        540588
    ],
    "type": [               (2)
        "144": [            (3)
            548066
        ]
    ],
    "tenants": [            (4)
        "497147": [         (5)
            "globals": [    (6)
                548169
            ],
            "type": [       (7)
                "144": [    (8)
                    496932
                ]
            },
            "ids": [        (9)
                "496931": [ (10)
                    545972
                ]
            ]
        ]
    ]
]
```

- (1) Contains the list of scripts objects stored directly under "Overrides"
- (2) For each direct subfolder of "Overrides" that has a name starting by the letter "S" an entry is created in "type" map (2). The key of such entry is the target subtype (as specified in the subfolder's name) while the value is the list of scripts contained the aforementioned subfolder.

- (4) For each direct subfolder of "Overrides" that has a name starting by the letter "D" an entry is created in "tenants" map (2). The key of such entry is the tenant's DataID (as specified in the subfolder's name) while the value is the tenant OM configuration.
- (5) For each "tenant" subfolder a sub-Override Map is created (SOM). The structure of SOM is identical to the one of OM with the only difference that subfolders of a tenant subfolder having a name starting with the letter "D" are used in SOM for creating entries in the "ids" map.

Below an exemplar content of the Overrides folder

Name	ID	SubType
Overrides	00001	AnsTemplateFolder
- GlobaScript	00002	Content Script
- S144	00003	Content AnsTemplateFolder
- - Document Script	00004	Content Script
- D1234	00005	AnsTemplateFolder
- - S0	00006	AnsTemplateFolder
- - - Folder Script	00007	Content Script
- - D5678	00008	AnsTemplateFolder
- - - Node Script	00009	Content Script

and the resulting OM

```
[
    "globals":[
        00002
    ],
    "type": [
        "144": [
            00004
        ]
    ],
    "tenants": [
        "1234": [
            "globals": [ ],
            "type": [
                "0": [
                    00007
                ]
            },
            "ids": [
                "5678": [
                    00009
                ]
            ]
        ]
    }
]
```

All Enhancements in version 3.0.0¶

ID	Scope	Description
#001130	Smart Pages	Add redirect and Smart View navigation capabilities to Smart Pages Controller script
#001119	Smart Pages	Added Iterator widget to Smart Page
#001120	Smart Pages	Added Include SmartPage widget to Smart Page
#001122	Beautiful Webforms	Two new uses cases for ADN
#001015	Module Suite	Content Script Performances improvements
#001097	Beautiful Webforms	Graphical request: item reference popup style with Smart View template
#001052	Smart Pages	Unable to access Content Script and some components with X-Content-Type-Options HTTP Header
#000990	Beautiful Webforms	Add 'Advanced customizations' configuration tab to the 'Custom Action Button' widget
#000672	Content Script	Getting nodes when a parent is a associated volume
#000993	Extension - Docx	Improved support for OpenDope custom XML Parts
#000624	Content Script	Being able of creating EMAIL object (subtype 733)
#000714	Content Script	Content-Disposition handler in Content Script
#000700	Beautiful Webforms	Retrieve Pulse comments

Issues Resolved in version 3.0.0¶

ID	Scope	Description
#001090	Online Documentation	Review a little detail in Workflow routing page
#001060	Content Script	Problem with AmWorkID and AMSubWorkID with form is status of a workflow
#001103	Smart Pages	Issue on the buttons of the Buttons Group widget (Smart Page)
#001104	Beautiful Webforms	Issue on the buttons of the Buttons Group widget
#001080		

ID	Scope	Description
	Online Documentation	Rend page: missing property and problem with Linux instruction (or in the package)
#001037	Content Script	Content Script: managecallbackso.cs is used and fails on an environment based on PostgreSQL DB
#001108	Online Documentation	Docx issue with Office 365 document
#001053	Content Script	managecallbacksm.cs script fail on a case sensive DB
#000891	Beautiful Webforms	Inconsistent behavior for check-boxes when used with Widget Space Content
#001040	Beautiful Webforms	Regression 029: form server side object is not correctly initialized if some field has default value
#000994	Beautiful Webforms	ADN DropDown widget is not working
#001016	Beautiful Webforms	No error message when validation is in OnLoad or on PreSubmit
#000642	Beautiful Webforms	Unable to access API documentation for Remote WebForms feature form.amRemotePack
#001041	Content Script	Regression 029: nodes loaded through getChildren(Fast) APIs are not properly initialized when versionables
#000944	Content Script	Document generated with a merge is corrupted if there are comments in the documents
#001034	Smart Pages	Form with Wysiwyg widget on Smartpages: dropdown menu and pop up for insert object are not showed properly
#001030	Smart Pages	Two small anomalies with Content Scripts in Smart UI: error in move operation and no way to see permissions
#001025	Content Script	Error checking attributes starting from a shortcut
#000985	Beautiful Webforms	Space Content: the uploaded document has random string in name
#001065	Beautiful Webforms	Radio selection reset after document upload
#001043	Content Script	Regression on patch 029: JDBC API is not working
#001094	Beautiful Webforms	Default in Modal Container
#001109	Smart Pages	

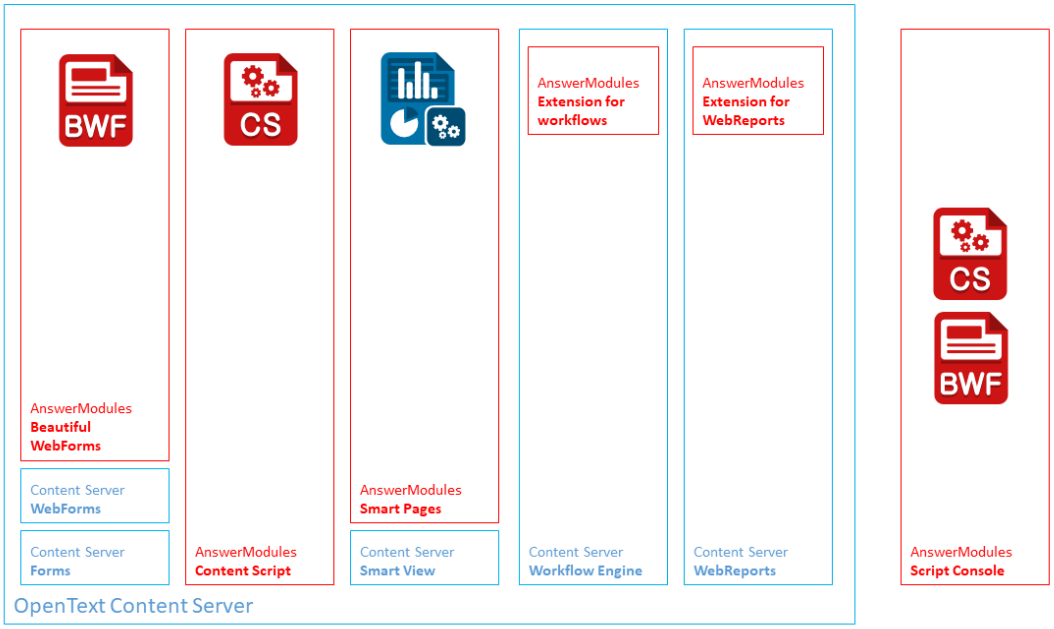
ID	Scope	Description
		CSSmartUIService is unable to deserialize page model if model.data contains classes that have been defined within a Script
#001056	Content Script	Regression on patch 029: timeout putting a value in cache
#000644	Beautiful Webforms	It is not possible to save an empty content script
#001028	Extension - xECM	Missing 'Inline Guide' for xecm extension
#001029	Extension - xECM	Wrong parameters type of editor autocomplete of the 'AddRole' method of the 'xecm' extension
#000939	Smart Pages	Erroneous behavior when selecting rows in Smart Pages Datatable widget
#000521	Beautiful Webforms	Source Code editor within Form Builder is initialized with wrong code when a new empty BWF view is created
#000998	Beautiful Webforms	Minor error in panel container
#001095	Beautiful Webforms	Scroll relocator: if added to a page there is a JS error
#001121	Beautiful Webforms	Error getting menu from a document
#000905	Beautiful Webforms	Datatable widget doesn't support client side actions (like pagination, search and sorting)
#000983	Beautiful Webforms	Multiple input field overlap date picket
#001038	Smart Pages	Missing search on columns in Node Table Table Tile
#001019	Beautiful Webforms	Existing Datatables widgets have data loading issues after applying hoftix_2.9.0_001
#000886	Smart Pages	Toggle Preview not available on Smart Page
#001000	Beautiful Webforms	Plus button not clickable on FireFox
#000957	Smart Pages	Widget Nodes table - Error on selecting nodes
#000971	Beautiful Webforms	Select from list widget ignore the selected value when it is in a tab
#000953	Beautiful Webforms	Workflow comment added many times with SmartView Template when Tab Action Buttons widget is used
#0001051	Content Script	

ID	Scope	Description
		Real fields in categories are assigned Float values if accessed through GCSPrimitiveAttribute
#000995	Beautiful Webforms	Model properties are not updated for widgets in layout containers.
#000991	Beautiful Webforms	Make library update more robust
#001013	Module Suite	ScriptManager Initialization invalidates Session Cache
#000980	Smart Pages	Custom columns created with new CSSmartView:Columns functionality not showing in Smart Views

Architecture

Module Suite

Module Suite for Content Server by AnswerModules is a comprehensive framework that includes various innovative solutions and extensions modules for OpenText Content Server.



Beautiful WebForms¶

The Beautiful WebForms Framework is an enhancement to the standard OpenText WebForms module that provides developers with all the required tools to create and manage next generation form based applications on Content Server. The module significantly contributes in delivering to the application's end users a modern, comfortable and ergonomic usage experience while at the same time lowering overall development and maintenance costs.

Content Script¶

Content Script is the first genuine scripting engine for OpenText Content Server. Content Script enables the creation of a new type of executable script object, capable of both automating actions that can be performed through the standard Content Server UI, as well as creating custom interfaces, consoles, reports, and more.

Content Scripts are foundation blocks that can be used to create any sort of application based on OpenText Content Server.

Note

Content Script API and API Extension Packages (CSEPs)

One of the most powerful features of Content Script lies in the fact that within the Content Script code it is possible to interact with Content Server itself and with external services or data sources through a set of service APIs. The API layer is engineered for extensibility, and new APIs are released periodically to enable the most various tasks. Also, thanks to the Content Script SDK, Modules Suite owners and developers can create their own extensions. CSEP can be enabled and disabled dynamically from within the administrative pages of Content Server.

Smart Pages¶

Smart Pages is a solution that allows developers to leverage the Content Script template engine's capabilities to create UI elements of any sort by adopting a rigorous MVC *(https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller)* design pattern. Smart Pages have been primarily engineered to be utilized in the context of Smart View applications, where they can be useful for creating Smart View perspective tiles. Smart Pages replaces the Module Suite View extension for Smart View which has been discontinued at version 1.8.

Script Console¶

Unlike the Content Script Module and Beautiful WebForms Module, which are standard extension modules and live inside OpenText Content Server, the Content Script Console is a standalone, multi-platform (Unix, Windows) environment for the execution of Content Scripts and Beautiful WebForms. As such, it is executed separately from Content Server, potentially on different physical environments (such as an Administrator's own workstation or a server in a network DMZ), but retains the capability of interacting with one or more Content Server environments.

Module Suite default extensions¶

Module Suite comes out-of-the box with a set of extensions that enable new usage scenarios for core Content Server modules.

Content Script Extension For Workflows¶

The Content Script Extension for Workflows allows you to add Content Script Steps to new or existing Content Server Workflow Maps.

Content Script Steps are automatic steps that will execute the associated Content Script when triggered. The execution outcome will be interpreted by the step itself in order to route the

Workflow to the next step. It is possible to build expressions that check for successful execution, execution errors or that interpret the outcome of the script.

The usage of Content Script Steps can reduce to a minimum the need for custom Event Trigger Scripts.

Content Script Extension For WebReports¶

The Content Script Extension for WebReports improves standard WebReports functionality by introducing new usage scenarios, such as:

- the possibility to use a Content Script as a Data Source for WebReports
- the possibility to execute WebReports from within a Content Script
- the possibility to execute Content Scripts from within a WebReport thanks to a custom subtag

Module Suite Extension For ClassicUI¶

The Module Suite Extension for CalssicUI is a simple and fast way to enhance the OpenText Content Server user experience.

This powerful tool gives the possibility to manage: - An objects menu options - Manage default and custom columns at run-time - Redesign guis by embedding fancy widgets - Customize the way items are being created in the system - Dynamically create forms without having to write HTML code - Easily perform massive operations

Module Suite Extensions

ModuleSuite Extensions enhance the capabilities of existing standard Content Server Modules, if they are installed on the systems.

ModuleSuite Extension For DocuSign¶

ModuleSuite Extension For DocuSign has been developed in order to dramatically simplify the integration between OpenText Content Server and the DocuSign® signing platform. These integration solutions are based on AnswerModules' core solution, Module Suite, and thanks to their outstanding flexibility can be utilized to implement all sorts of use-case scenarios.

Most common usage scenarios

- Manually starting a DocuSign® signing workflow directly within the Content Server UI in order to have a set of Content Server documents signed by a group of Content Server users
- Manually starting a DocuSign® signing workflow directly within the Content Server UI in order to have a set of Content Server documents signed by a group of external users;
- Managing one or more DocuSign® signing workflows, each one involving both Content Server users and non-Content Server users, as part of the execution of a Content Server internal workflow

ModuleSuite Extension For ESign¶

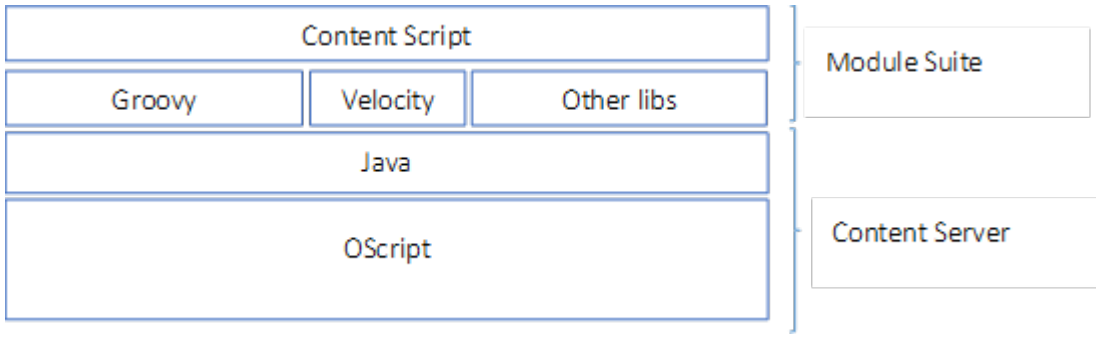
ModuleSuite Extension For ESign allows for Beautiful WebForms to be used as the signing step in a signature workflow.

Applicative Layers

One of the main reasons that brought to the creation of the Module Suite was the need to improve Content Server's capability of integration with other systems. For this very reason, on top of an OScript Layer that implements most of the Content Script Core functionalities, we developed an integration layer that makes use of the Content Server embedded Java Virtual Machine.

Content Script was developed with a language grammar and syntax fully compatible with Groovy, the well-known Scripting language for Java, in order to speed up development and most importantly open Content Server to a wider range of developers than the reduced OScript developers' community.

On the other hand, being OScript's grammar very similar to Groovy's, OScript developers should easily find their way with the Content Script language.



Note

In recent years, more and more functionalities of Content Server have been making use of the embedded Java Virtual Machine. Nevertheless, the standard level of isolation of these components has not yet been significantly improved. It is still up to system administrators and developers to manually assure the absence of conflicts in the system when new Java libraries become necessary. Module Suite comes with a higher level of isolation and implements its own additional libraries management

Requirements, links and dependencies

Module Suite Compatibility Matrix¶

OpenText Content Server	MS 3.2.1	MS 3.3.0	MS 3.4.0	MS 3.5.0	MS 3.6.0	MS 3.7.0
Content Suite 21.1	X	X	X			
Content Suite 21.2	X	X	X			
Content Suite 21.3	X	X	X			
Content Suite 21.4	X	X	X			
Content Suite 22.1	X	X	X		X	X
Content Suite 22.2	X	X	X		X	X
Content Suite 22.3		X	X		X	X
Content Suite 22.4			X		X	X
Content Suite 23.1			X(*)		X	X
Content Suite 23.2				X	X	X
Content Suite 23.3				X	X	X
Content Suite 23.4					X	X
Content Suite 24.1					X(**)	X
Content Suite 24.2						X

Dependencies¶

Module or Component	Included In	Depends On
Content Script	-	-
Beautiful WebForms	-	Content Script
Smart Pages	-	Content Script
Script Console	-	Content Script

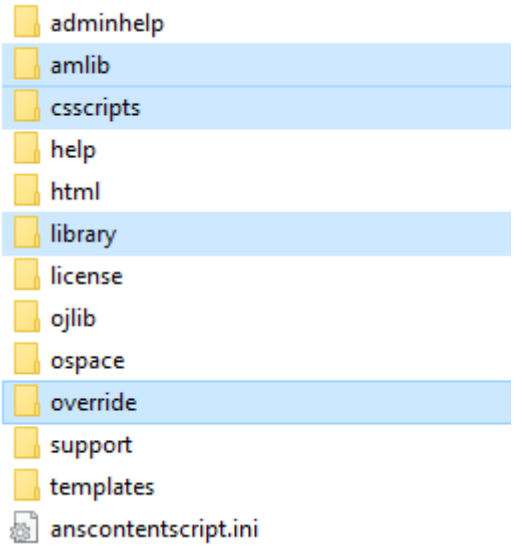
Module or Component	Included In	Depends On
Remote Beautiful WebForms	Script Console	Beautiful WebForms
Module Suite Extension For WebReports	Content Script	WebReports
Module Suite Extension for Workflows	Content Script	
Module Suite Extension for Classic UI	AMGUI Ext.Pack	Content Script
Module Suite Extension for ESign	AnswerModules Content Script eSign ExtPack	Content Script, Beautiful Webforms, ESign
Module Suite Extension for DocuSign	AnswerModules Module Suite extension for DocuSign	Content Script, Beautiful Webforms, Script Console

Modules layouts

Module Suite's modules present a peculiar layout that differentiate them from most of the Content Server's modules you might have worked with. Knowing the modules' internal structure is of primary importance when it comes to: upgrading, maintaining or extending your Module Suite instance.

Content Script¶

Content Script features a set of layout differences in respect to standard Content Server modules. In the following paragraphs each one of these differences is discussed in details.



amlib¶

The “**amlib**” directory contains all the core libraries of the Content Script Java Layer. It is also used to deploy and manage Content Script Extension packages. If a Content Script API Service (CSAS) , made available from a CSEP, needs to load its own Java libraries, then they will be deployed in a sub-directory of the amlib directory having the same name of the Content Script API Service identifier. This way, two different Content Script API Services can load two different version of the same Java library.

csscripts¶

Content Script scripts can be used and also invoked directly from OScript. Scripts under this folder can be executed as part of OScript scripts or functions. Some of them are used to implement Module Suite’s administrative pages.

> The Content Script OScript APIs are not covered by this training manual.

library¶

Module Suite's components behaviour and functionalities can be modified and extended by manipulating the content of the **Content Script Volume** (a Content Server’s Volume created when installing the Content Script module).

The purpose of most of the structure and content of the Content Script Volume can be easily understood by simply navigating the volume thanks to the "convention over configuration" paradigm that has been adopted. That means that most of the time, simply creating the right Content Script, Template Folder or Template in the right place will be enough to activate a specific feature. The default configuration (i.e. the default Content Script Volume's structure) should be imported as part of the installation procedure of the Content Script module.

In the next sections we will refer to specific locations in the Content Script Volume content as "**Component Library**" or simply "**Library**". This directory contains the default initial version of the Library and will be used later on to manage Library’s backups and upgrades. The Library can always be imported, exported or upgraded directly from the Module Suite’s administrative pages.

override¶

Content Script can be used to deeply customize the Content Server standard UI through a non-disruptive (applying non-permanent modifications) functionality that allows developers to override the standard result of a Content Server **weblingo** file evaluation with the result of a Content Script execution.

Weblingo override functionality is controlled by XML configuration files to be placed in the " *override*" folder in the anscontentscript module.

```
<?xml version="1.0" encoding="UTF-8"?>
<override>
    <active>false</active>
    <target>
        <![CDATA[E:\OTHOME\module\webattribute_10_5_0\html\attrstring.html]]>
    </target>
    <!-- Content Script ID -->
    <script>ID</script>
    <!-- BEFORE, AFTER, CUSTOM -->
    <mode>CUSTOM</mode>
    <!-- Optional Script's Parameters -->
    <params>
        <entry>
            <key>key</key>
            <value>value</value>
        </entry>
    </params>
</override>
```

Within the folder, you should find a sample XML configuration file that should be quite self-explanatory. The XML file points to a Content Script object, identified by *dataID*, which implements the functionality.

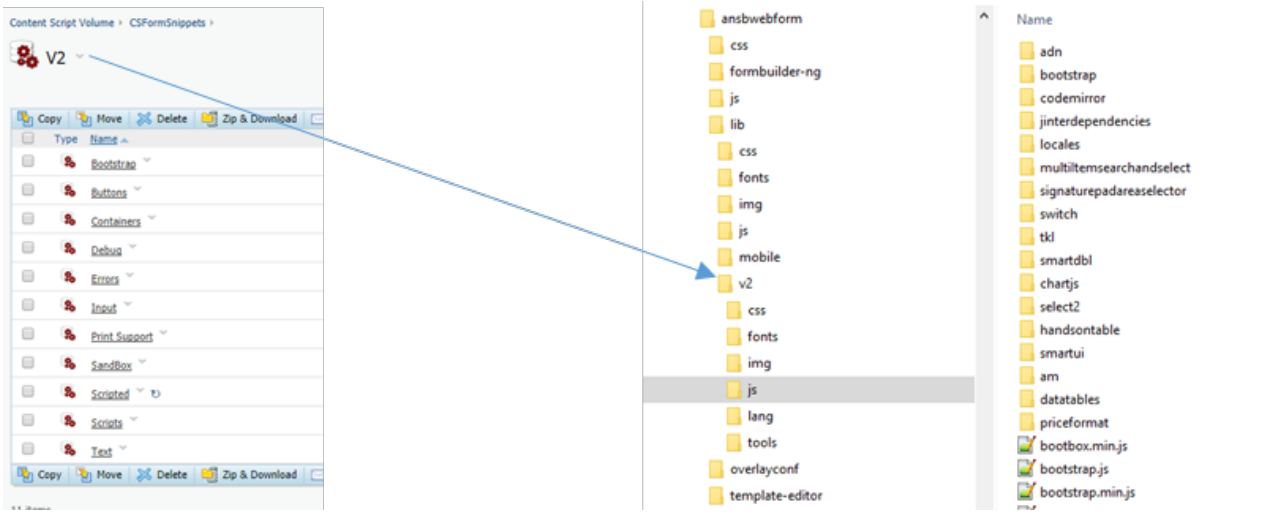
Setting the "*active*" flag to "*true*" will activate the override.

> Please note that this is a very low-level functionality and such might have a significant impact on users' experience ***use it with caution***. The feature requires a restart every time the configuration is changed.

Beautiful WebForms¶

The most relevant aspects of the module's internal structure for the Beautiful WebForms module are related with the "**support**" directory. Beautiful WebForms default View Templates make use of several JavaScript libraries: they have been selected, written and optimized to work together with View Templates.

In particular, the Beautiful WebForms' unique validation framework makes use of the libraries stored under the "**js**" directory. The recommended way to load these libraries is to make use of the Velocity macros expressly designed to load them

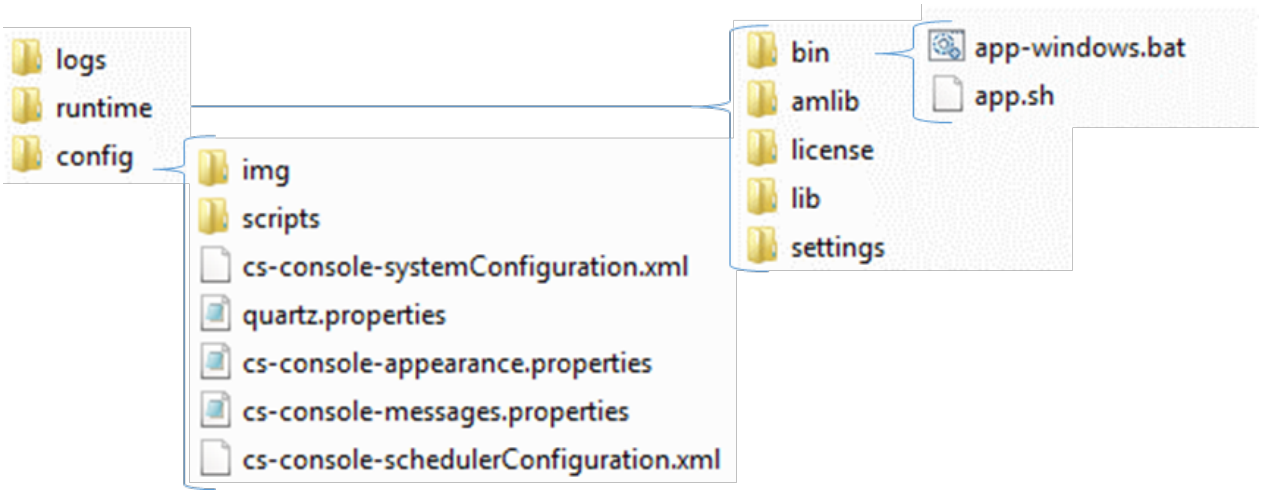


Starting with version 2.0 the module's static resources have been deeply revised and re-organized. They are now structured in a way that reflects the way Beautiful WebForms' widgets are organized in the Content Script volume. Beautiful WebForms' widget are in fact now organized into libraries.

Script console¶

The Script Console internal structure reflects its ability to connect to multiple Content Server Instances and to organize Content Script scripts in multiple repositories.

Since version 1.7.0, the Script Console runtime and configuration folders are all stored under the same installation path. The Script Console installation folder will appear as shown in figure here below:



Script Console main configuration file¶

The Script Console main configuration file (**cs-console-systemConfiguration.xml**) is stored under the **config** directory. As the naming of the file tells us, it is an XML based configuration file, intended to include general configuration parameters of the Script Console as well as

specific settings related to the Content Server system to which the Script Console can be connected.

The configuration file is automatically modified by specific actions performed on/through the Console (such as registering a new target Content Server system) or can be edited manually by the administrators.

Installation and Upgrade

Installing Module Suite

installation

Getting ready to install Module Suite¶

Overview of the Module Suite installation process¶

This guide describes the step-by-step procedure that will lead to the installation of the Module Suite on a Content Server environment, including the following components:

- Content Script
- Beautiful WebForms
- Smart Pages

Install Module Suite components separately

If you are only interested in installing a subset of the available components, please check the dedicated installation guides for additional guidance:

- Installing Content Script
- Installing Beautiful WebForms
- Installing Smart Pages

Script Console installation

Script Console is a special component that is part of the Module Suite product but follows a different deployment pattern. This guide does not cover the installation of Script Console.

If you are interested in the Script Console installation, please check the following guide: Installing Script Console.

Depending on the characteristics of the target environment (Unix/Linux or Windows, single server or clustered, ...) different options might be provided for each installation phase.

The following high-level phases will be covered:

1. Deployment
   This phase covers the deployment of the software binaries on the target system. The related operations will be typically performed with a click-through installer.

2. Installation
   This phase covers the "installation" phase of the deployed Modules within the target

Content Server system. The operation is performed through the standard OpenText Content Server Administration tools.

3. Activation
   This phase covers the available procedures to apply the required software keys and activate the Module Suite software. The operations are performed using AnswerModules Administration tools available within the Content Server Admin pages and standard OpenText Content Server Administration tools.

4. Configuration
   This phase covers the minimum set of post-installation configuration steps that are necessary to get the software up and running. This includes importing certain core libraries and components in the system. The operations are performed using AnswerModules Administration tools available within the Content Server Admin pages.

5. Post-installation patching
   From time to time, hotfixes and patches are released to provide new features and address product issues. It is always suggested to keep the system up-to-date with all relevant patches and hotfixes, starting from the initial installation.

Installing on a Clustered Environment

When installing on a clustered Content Server environment, the overall installation procedure will vary.

In a clustered environment it is **mandatory** to install the Module Suite components on all nodes, but it is important to notice that the single installation steps must not be performed on each single node separately, as certain operations already affect the whole cluster.

At a high level, the suggested procedure is to perform a complete installation on the primary node of the cluster, and then reconcile the remaining nodes.

Please refer to the Installing on a clustered environment guide for detailed info.

Prerequisites¶

This guide assumes certain resources to be readily available while performing the installation. Please ensure the following have been provisioned before starting the installation process:

- ☑ Admin-level access to the servers on which the software will be installed
- ☑ Admin user access to the Content Server instance.
- ☑ The Module Suite **installers** or installation packages compatible with the target environment

Installer versions

Before proceeding with the installation, make sure that the installer version matches the OpenText Content Server target system version.

E.g.:

- ***module-suite-2.7.0-OTCS162.exe*** is the Windows installer for OpenText Content Server 16.2.X;
- ***module-suite-2.6.0-OTCS162.exe*** is the Windows installer for OpenText Content Server 16.2.X;
- ***module-suite-2.5.0-OTCS162.exe*** is the Windows installer for OpenText Content Server 16.2.X;
- ***module-suite-2.4.0-OTCS16.exe*** is the Windows installer for OpenText Content Server 16.0.X;
- ***module-amcontentscript-2.3.0-OTCS105.exe*** is the Windows installer for OpenText Content Server 10.5.X;
- ***module-amcontentscript-2.2.0-OTCS10.exe*** is the Windows installer for OpenText Content Server 10.0.X;

Note: Starting with version 3.2.0, the OTCS identifier (OTCS10, OTCS105, OTCS162 ...) is no longer present in the installer names.

☑ A valid AnswerModules **activation key**, either in plain text format or in OTCS Configuration Export XML format. The latter is the suggested option as it will prevent errors due to manual input.

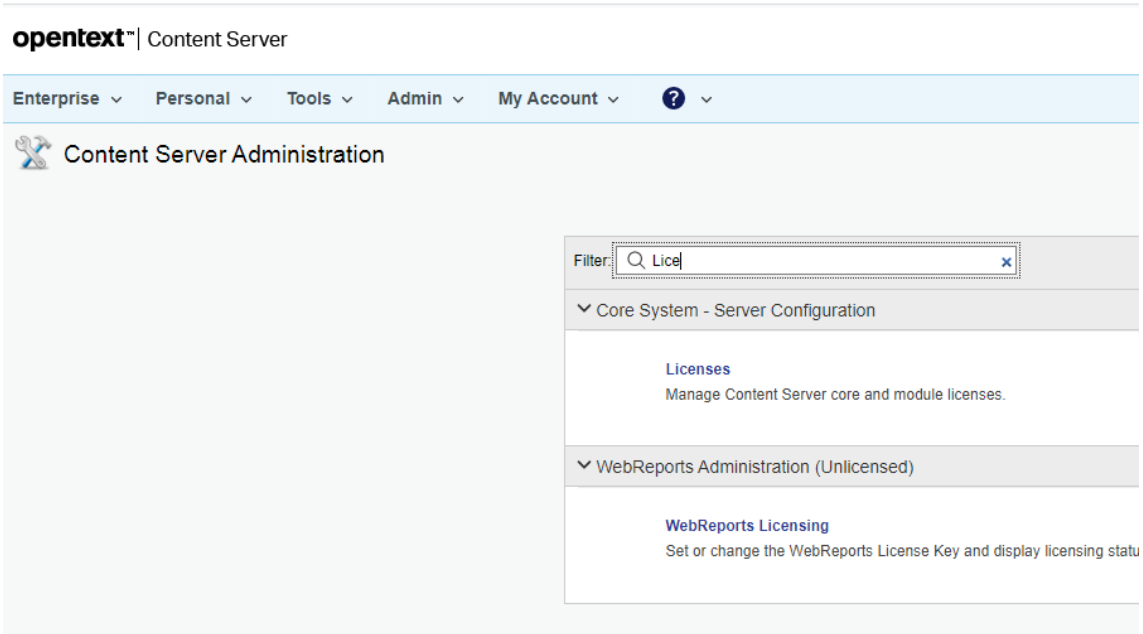
Keys and System Fingerprint

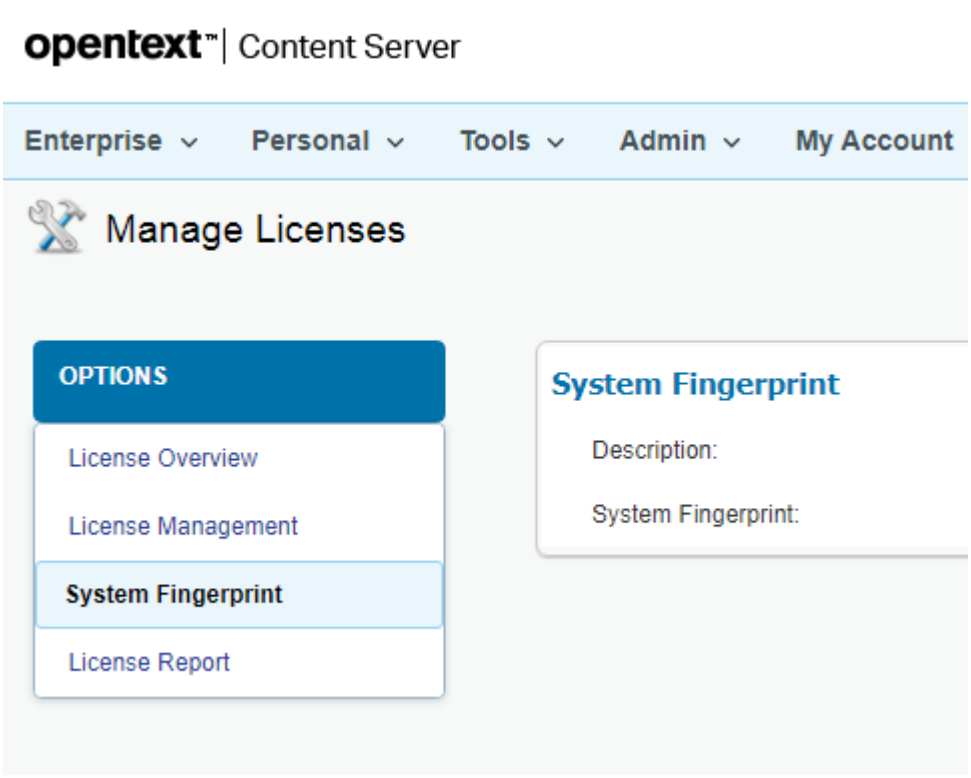
- An activation key is only required starting from version 1.7.0 of the Module Suite.
- Starting from version 2.0.0 activation keys are bound to the system's fingerprint.

How do I get an activation key?

In order to activate Module Suite you need a valid activation key. Activation keys can be requested to AnswerModules Support *(https://support.answermodules.com)* by providing the OpenText Content Server System Fingerprint.

You can read your's environment fingerprint from the OpenText Admin Pages as shown below





- [x] Any relevant **hotfixes** released for the Module Suite version being installed

Hotfixes

Hotfixes and patches are continuously published on the AnswerModules Support Portal. Check the availability of applicable patches when starting a new installation.

E.g. https://support.answermodules.com/portal/en/kb/articles/module-suite-3-5-0-hotfixes-and-patches *(https://support.answermodules.com/portal/en/kb/articles/module-suite-3-5-0-hotfixes-and-patches)*

Next Steps

Once all the prerequisites are met, please proceed to the **Deployment** phase:

- if you are installing on a Windows environment: Deploy on Windows
- if you are installing on a Unix/Linux environment: Deploy on Unix/Linux

Deploy

installation

Module Suite installation guide: Deploy Modules on Windows¶

Overview¶

This guide covers the **Deployment** phase that is part of the Module Suite installation guide.

- [x] **Deployment**
- [ ] Installation
- [ ] Activation
- [ ] Configuration
- [ ] Post-installation patching

This phase covers the deployment of the software binaries on the target system. The related operations will be performed with a click-through installer.

We will refer to the Content Server main installation directory as **%OTCS_HOME%** .

Platform specific

This guide is specific to the installation steps for a **Windows** environment. If you are installing on a **Unix/Linux environment**, please refer to Deploy on Unix/Linux.

Installers

The guide assumes that the required Module Suite installers for Windows have been provisioned and copied on the file system of the target environment.

Step-by-step Deployment¶

In order to deploy the Module Suite components, please follow these steps:

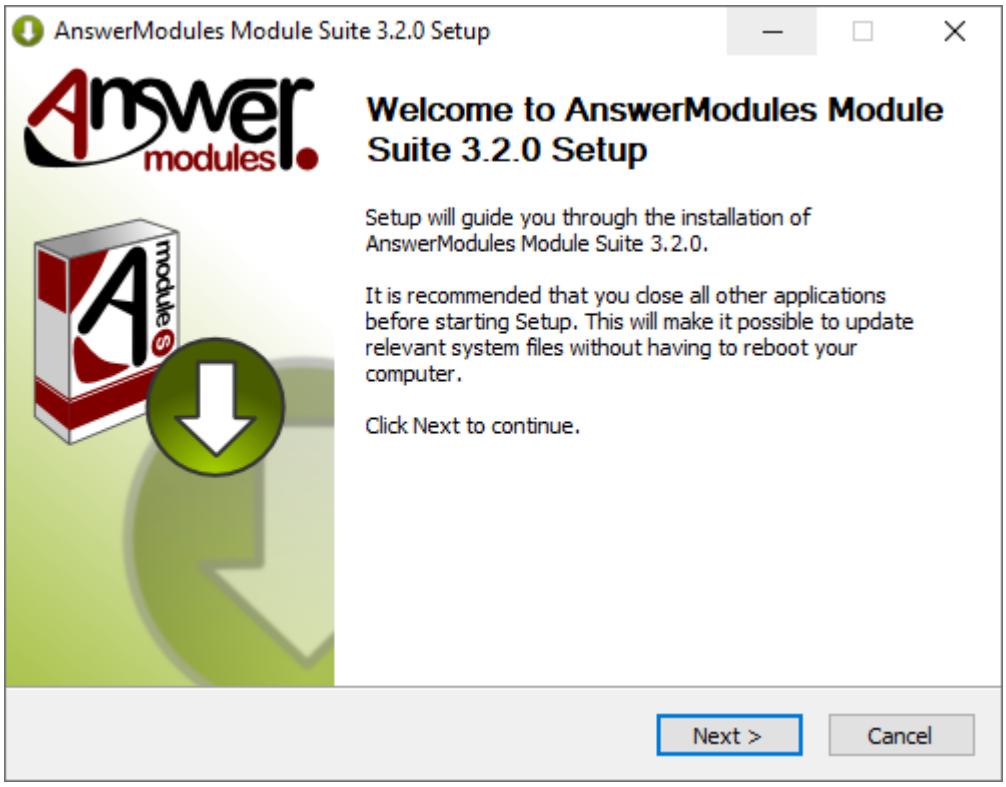
- [x] Stop the Content Server services
- [x] Run the **Module Suite Master Installer**

At this time, we will be installing the core Module Suite Content Server modules (Content Script, Beautiful WebForms, Smart Pages) and all the desired Module Suite Extension packages.

The following screens will guide you through the deployment of Module Suite modules.

- Welcome screen:

Select “Next” when ready to start the installation.



- Accept Module Suite EULA:

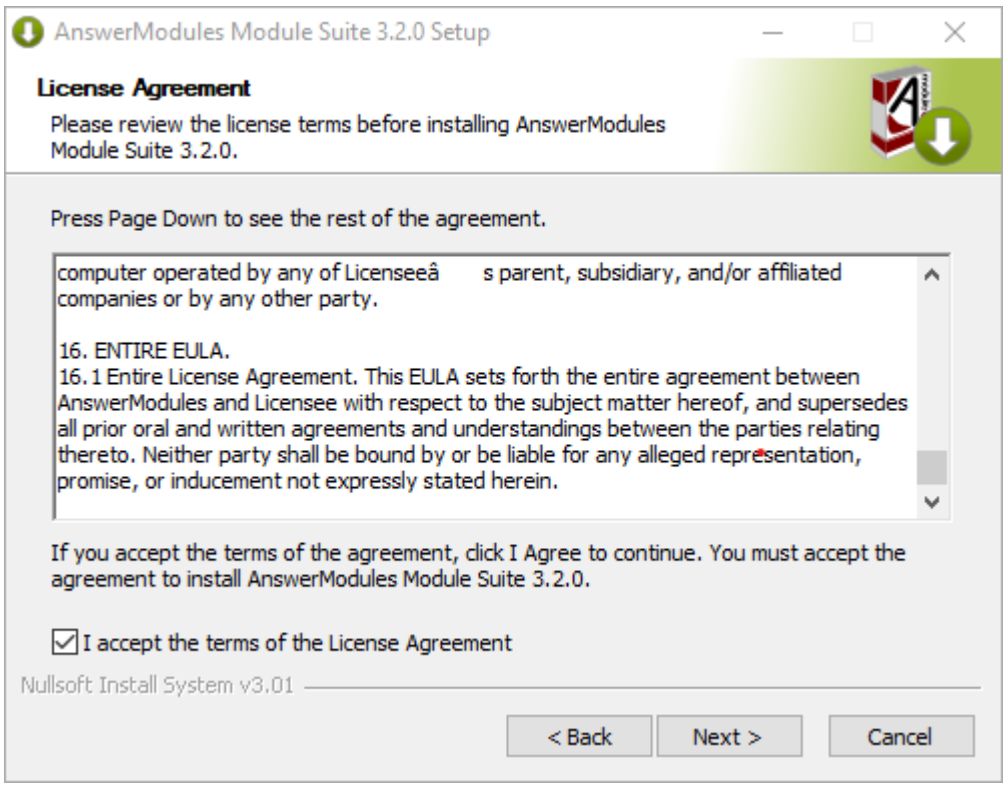
Acceptance of the end-user license agreement is mandatory for proceeding with the installation.

Accepted agreement

A copy of the EULA agreement will be available, upon installation, in:

%OTCS_HOME%/module/amcontentscript_X_Y_Z/license/EULA

Select “Next” when ready.



- [x] Select the components to be installed:

Select the components to install.

Partial installation

If you are intending to install only a subset of components, uncheck the elements that are not required from the list.

Dependencies

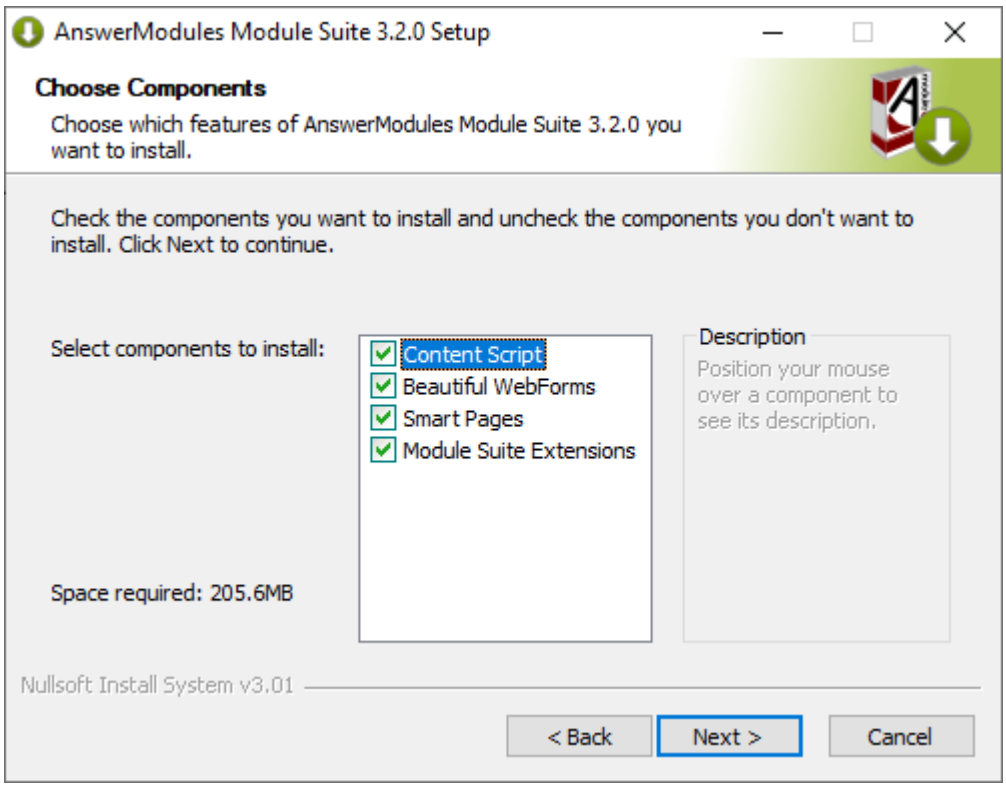
The following components:

- Beautiful WebForms
- Smart Pages
- Module Suite Extensions

depend on the "Content Script" module.

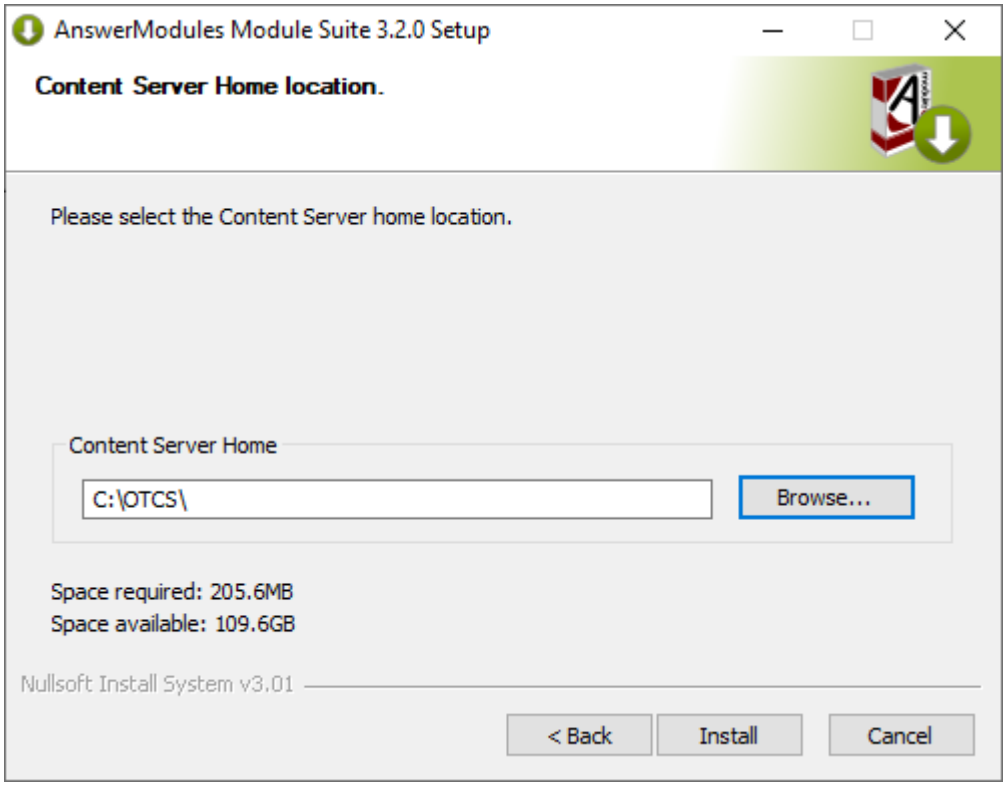
If you are intending to perform a partial installation, please make sure that "Content Script" is either selected or has already been installed in the system.

Select “Next” when ready.



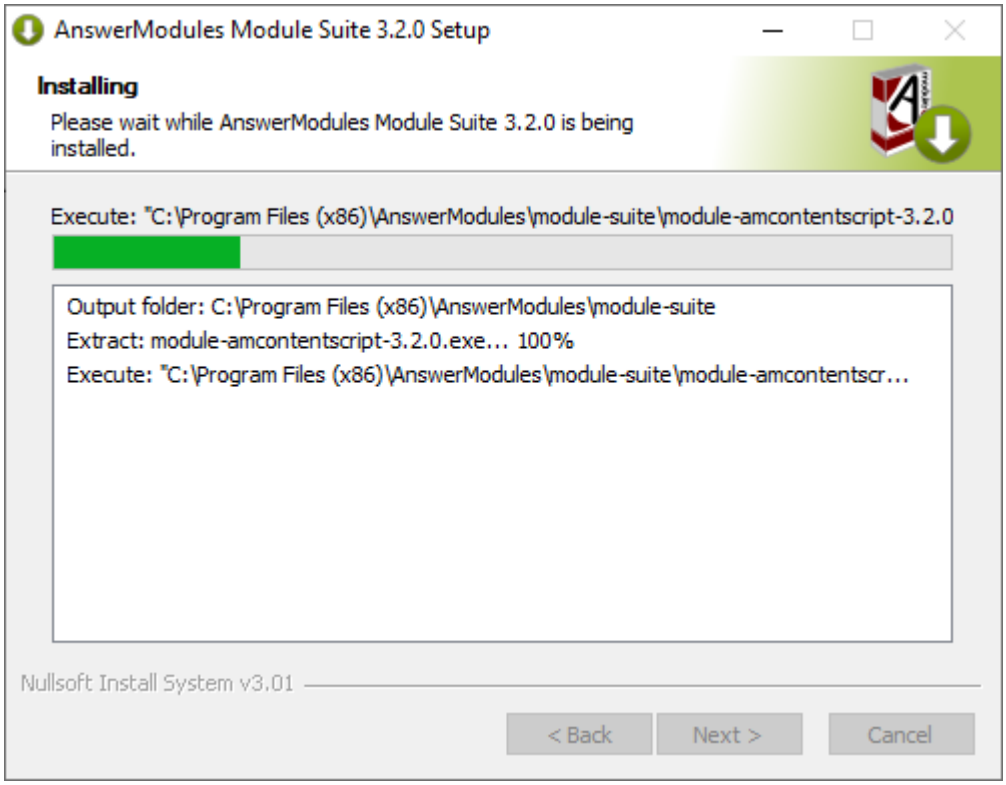
- [x] Confirm the installation path:

  The installer will prompt you for the location where Content Server is installed. Browse to your **OTCS_HOME** and select "Next" when ready to start the installation.



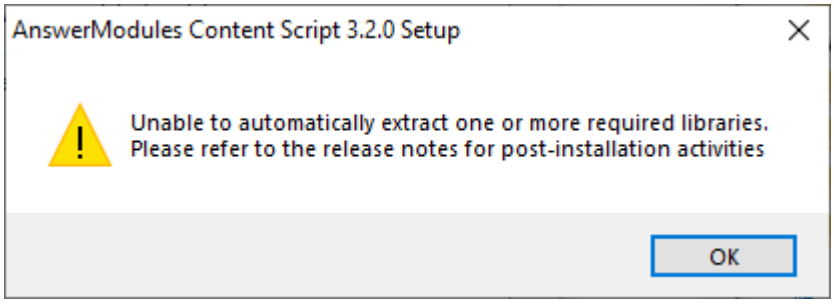
- [x] Deployment (automatic step):

Automatic import of Content Server dependencies: The installer will automatically attempt to load a few libraries from Content Server.
In case of failure, a warning message could appear during this phase of the installation. In such case, the operation must be performed manually.



What to do if the installer raises the error: Unable to automatically extract...

Some Content Script extension packages require two Java libraries that are specific to the target Content Server environment and are not distributed with the module.



The required library files are:

- csapi.jar
- service-api-X.X.XX.jar

and can be found in the web app located in:

- %OTCS_HOME%\webservices\java\webapps\cws.war (on CS 16.X)
- %OTCS_HOME%\webservices\java\webapps\cws.war (on CS 10.5.X)

To retrieve the files:

- copy the file named XXX.war to a temporary folder
- rename the file XXX.war in XXX.zip .
- extract the zip archive contents locate the files in the WEB-INF/lib folder

Once the files have been located, copy them to the folder:

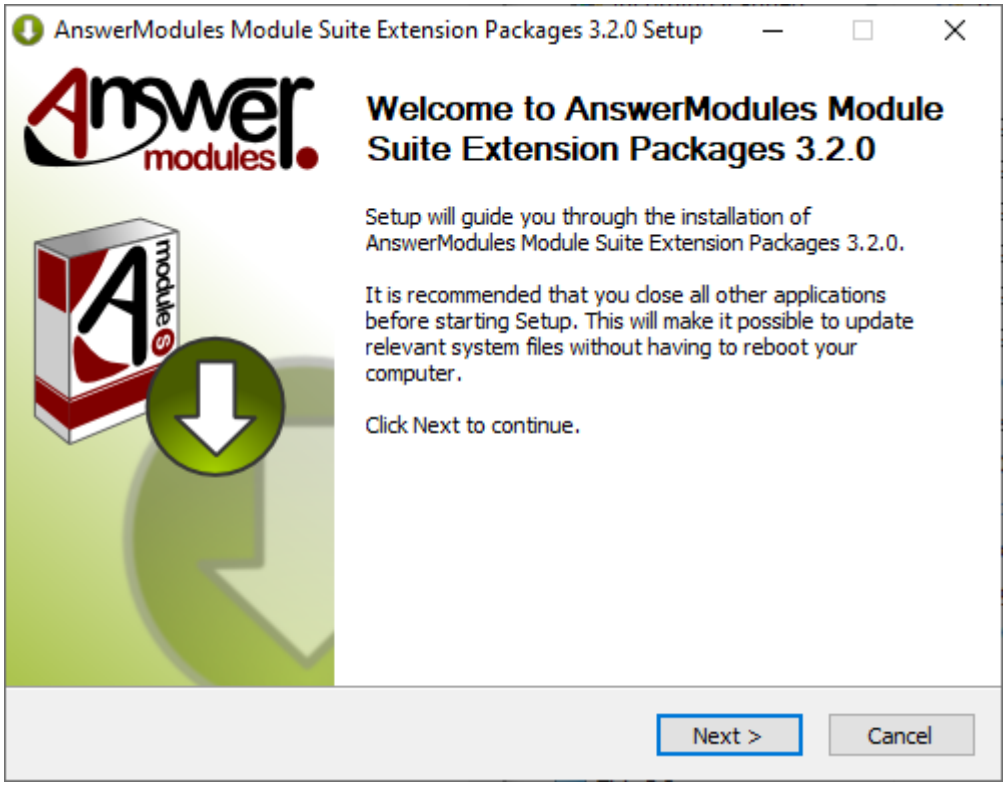
%OTCS_HOME\staging\anscontentscript_x_y_z\amlib

- [ ] Start the extension packages installation:

Optional

This will only appear if the "Module Suite Extensions" option has been selected in the master installer.

Welcome Screen: Select “Next” when ready to start the installation.



- [ ] Accept the extensions supplemental EULA:

Optional

This will only appear if the "Module Suite Extensions" option has been selected in the master installer.

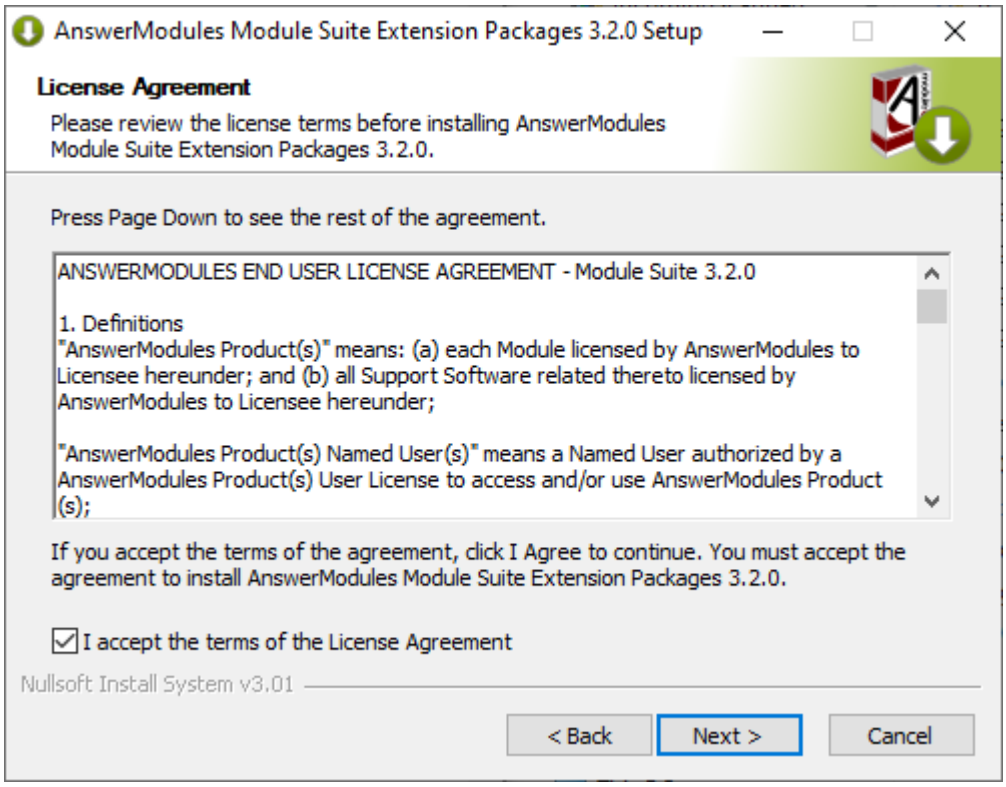
EULA Screen: Acceptance of the end-user license agreement is mandatory to proceed with the installation.

Accepted agreement

A copy of the EULA agreement will be available, upon installation, in:

%OTCS_HOME%/module/amcontentscript_X_Y_Z/license/EULA

Select “Next” when ready.



- [ ] Select the extension packages to be installed:

Mandatory Components

During the deployment phase, two components are mandatory and MUST be installed:

1. Module Suite Extensions Cache
2. Module Suite Extensions SQL

These components are prerequisites for several Administration tools, including the Content Script Volume Import tool.

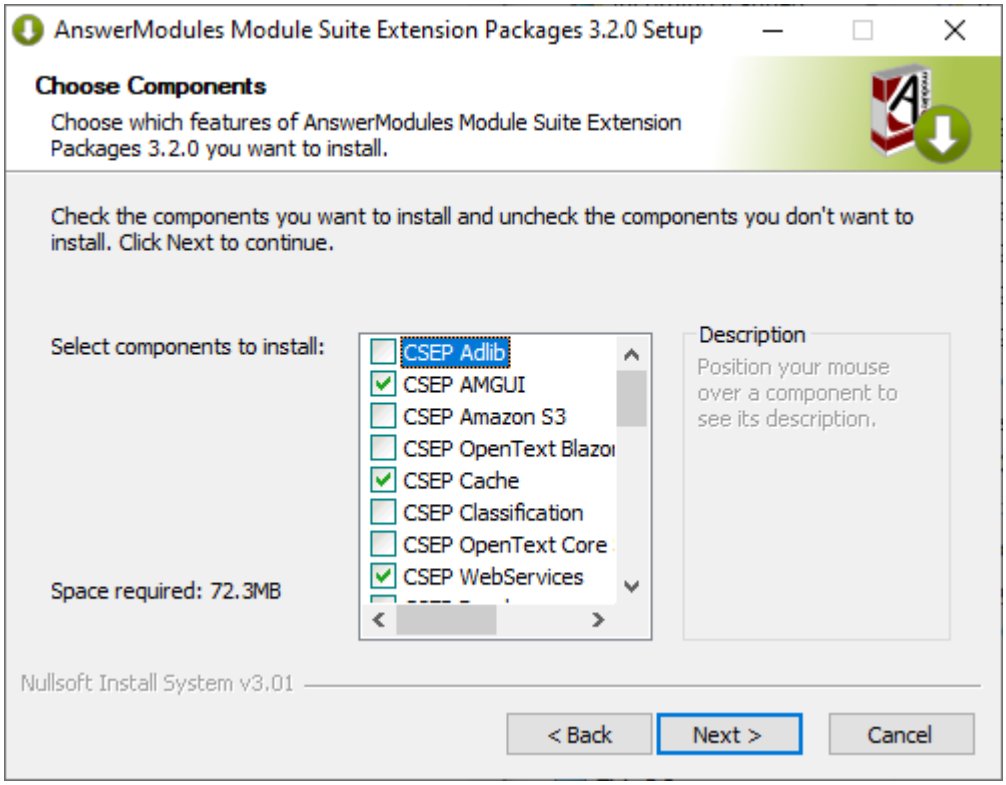
Do not skip mandatory components

Failing to install these components may result in certain Administration tools not functioning correctly.

Optional

This will only appear if the "Module Suite Extensions" option has been selected in the master installer.

Components selection: Select all of the extension components that are to be installed. Select “Install” when ready.



CSEP SAP

The Content Script Extension for SAP™ is a Content Script optional extension package that requires specific additional configuration steps.

It should not be deployed if you are not intending to complete the configuration, as an incomplete configuration could affect the Module Suite functionality.

This extension package requires the SAP™ JCo library *(https://support.sap.com/en/product/connectors/JCo.html)* to be available in the extension repository `<OTHOME>/module/anscontentscript_x_y_z/amlib/sap` and is certified for use with SAP™ JCo version (3.0.6) when used on OpenText Extended ECM and version (3.0.10) when used on CSP. SAP™ JCo library *(https://support.sap.com/en/product/connectors/JCo.html)* can be downloaded from SAP™ website.

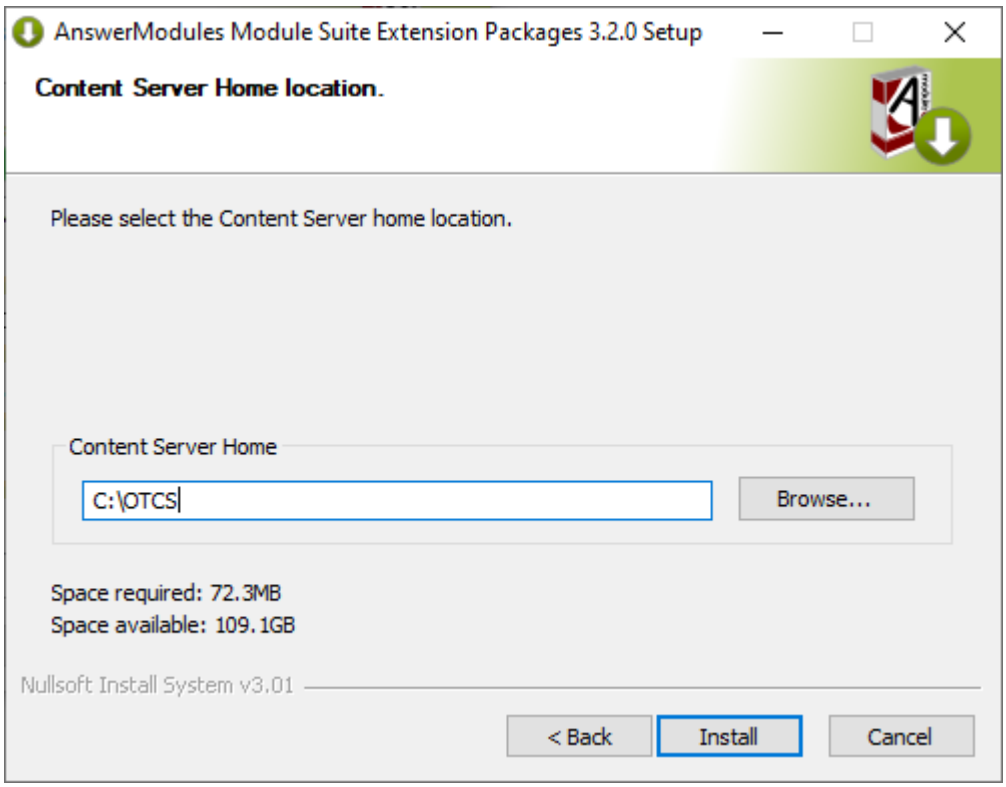
More on this extension.

- [x] Confirm the installation path:

Optional

This will only appear if the "Module Suite Extensions" option has been selected in the master installer.

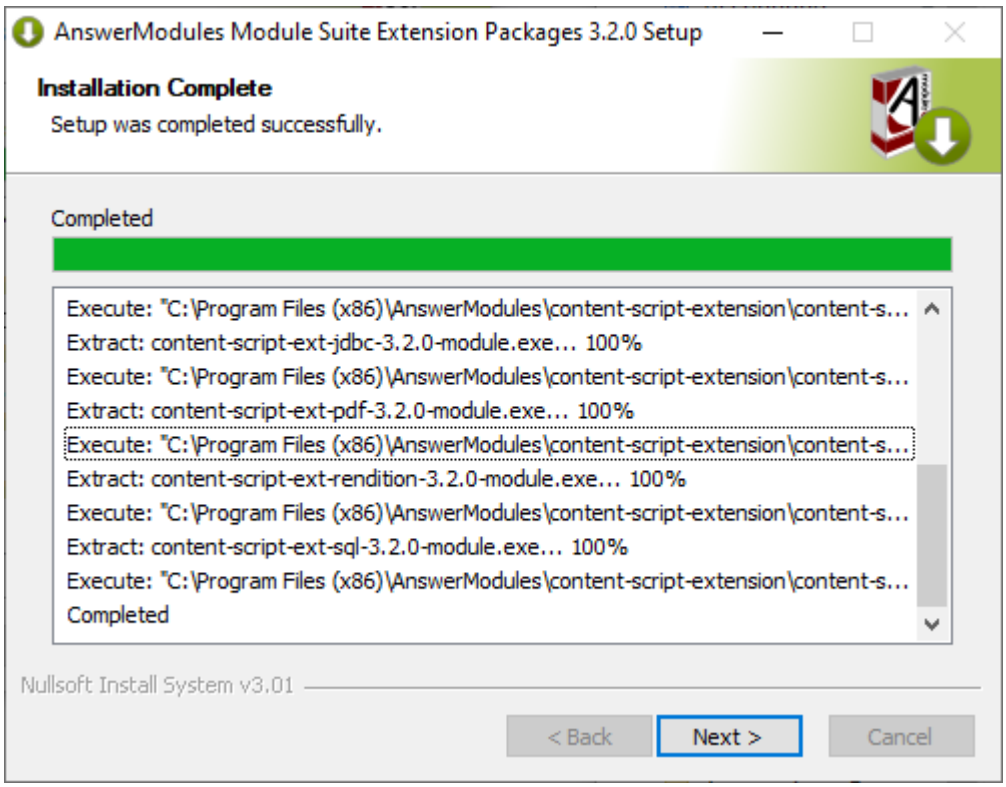
The installer will prompt you for the location where Content Server is installed. Browse to your **OTCS_HOME** and select “Next” when ready to start the installation.



- [x] Deployment (automatic step):

> **Optional**
>
> This will only appear if the "Module Suite Extensions" option has been selected in the master installer.

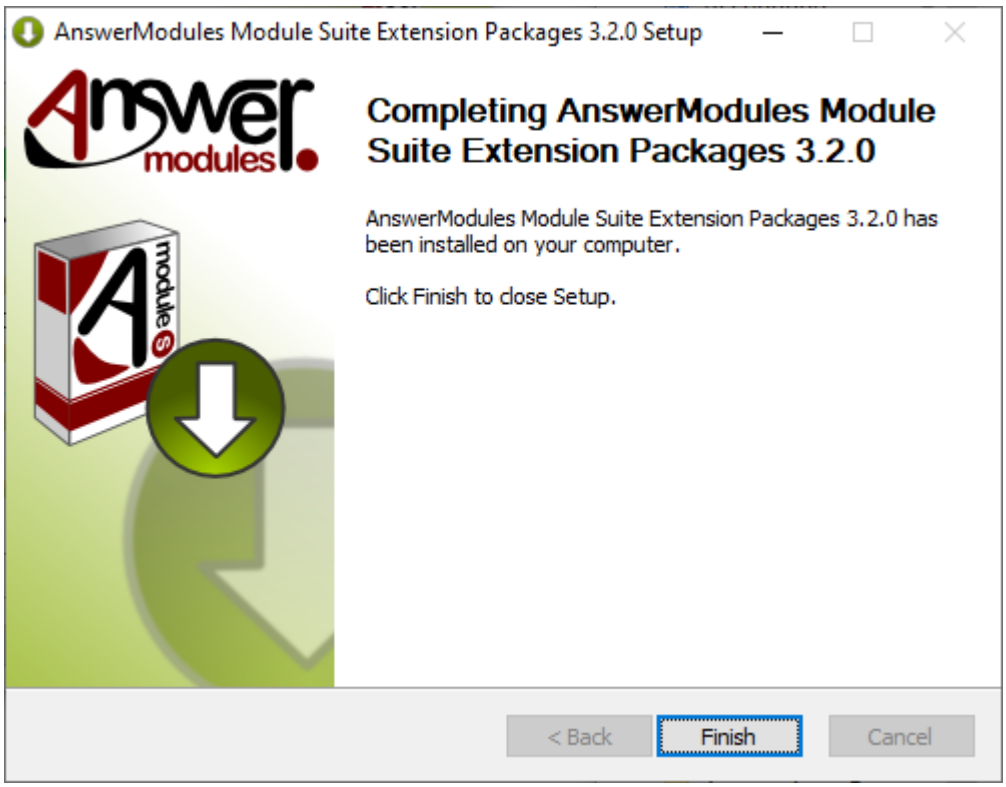
Extension Package Installation: The extension packages are automatically installed. Select “Next” when the procedure is complete.



- [x] Deployment complete:

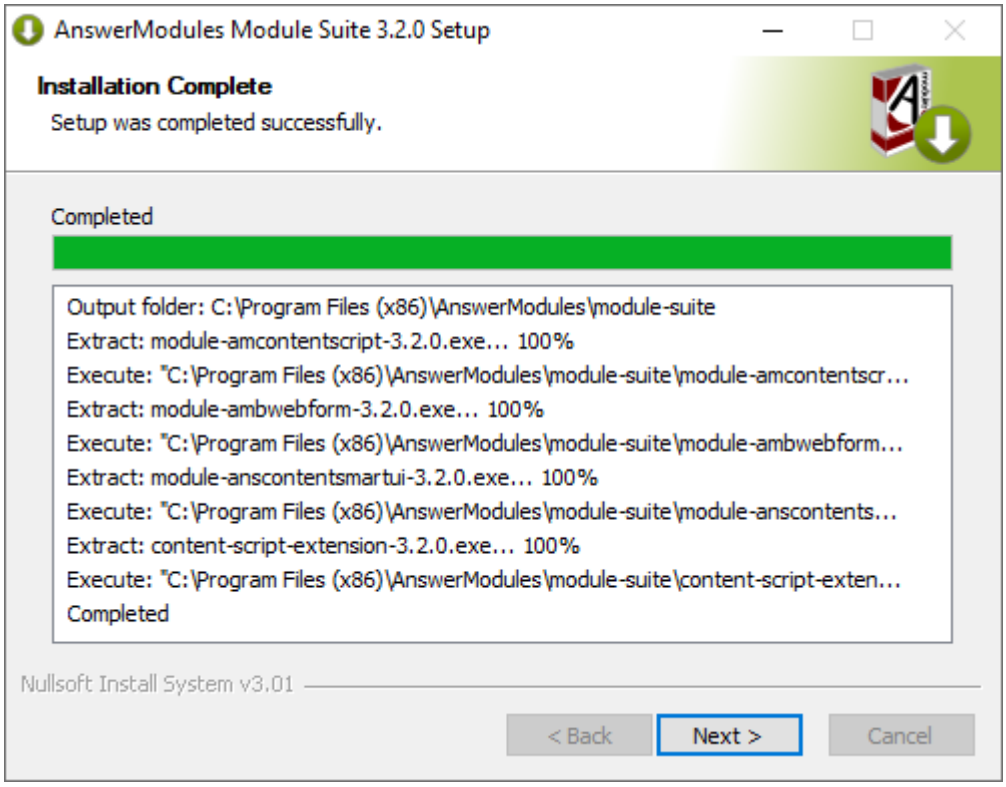
> **Optional**
>
> This will only appear if the "Module Suite Extensions" option has been selected in the master installer.

Extension Package Installation completed: Select “Finish” and return to the installation checklist to finalize the module setup.



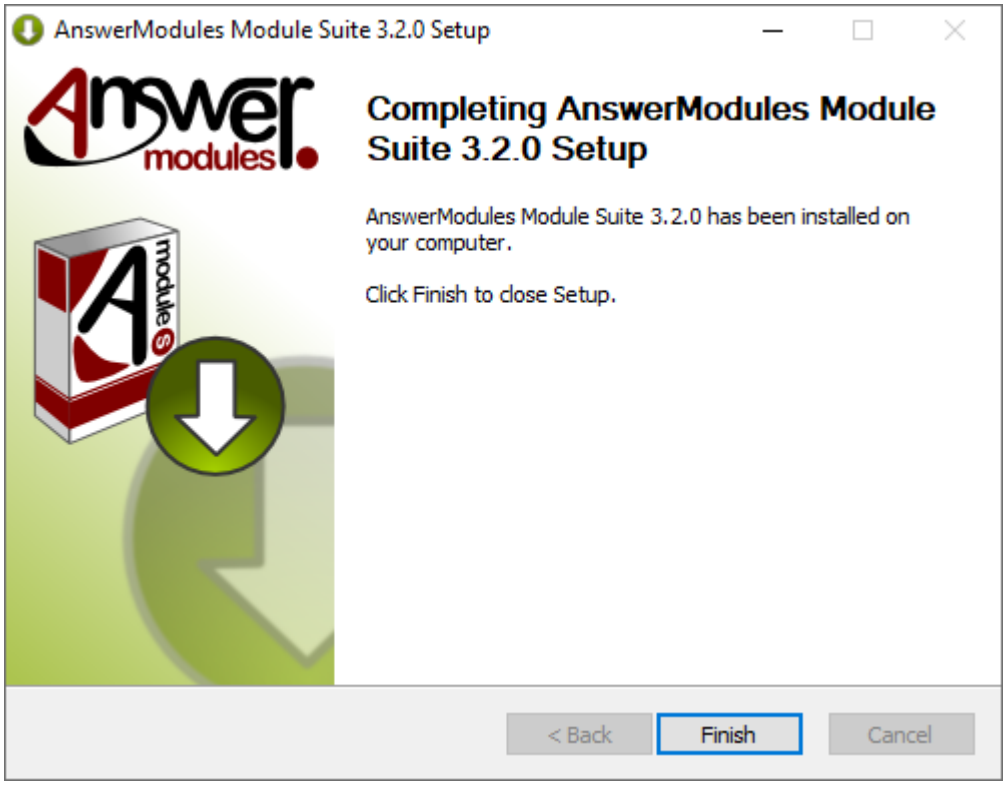
- [x] Master installer deployment (automatic step):

  Module Suite Installation: Module Suite components installation is finalized. Select “Next” when the procedure is complete.



- [x] Deployment complete:

Module Suite Installation completed: Select “Finish” and return to the installation checklist to finalize the module setup.



- [x] At this point, the Modules have been deployed in the Content Server Staging folder and is available for installing it through the Content Server administration pages.

Next Steps

Please proceed to the Installation phase.

installation unix

Module Suite installation guide: Deploy Modules on Unix/Linux¶

Overview¶

This guide covers the **Deployment** phase that is part of the Module Suite installation guide.

- [x] Deployment
- [ ] Installation
- [ ] Activation

- [ ] Configuration
- [ ] Post-installation patching

This phase covers the deployment of the software binaries on the target system. The related operations will be performed with a click-through installer.

We will refer to the Content Server main installation directory as **%OTCS_HOME%** .

Unix/Linux expertise required

This guide assumes a good working knowledge of a Unix System and its commands

Platform specific

This guide is specific to the installation steps for a **Unix/Linux** environment. If you are installing on a **Windows environment**, please refer to Deploy on Windows.

Installers

The guide assumes that the required Module Suite installer scripts for Unix/Linux have been provisioned and copied on the file system of the target environment.

Step-by-step Deployment¶

In order to deploy the Module Suite components, please follow these steps:

- [x] Stop the Content Server services
- [x] Open a terminal window

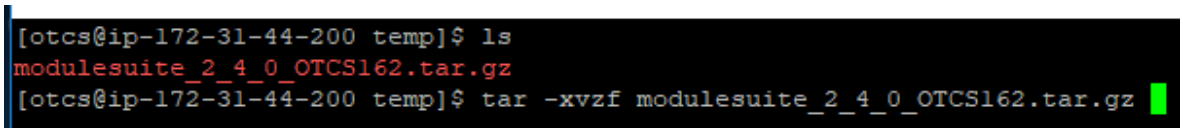
  At this time, we will be installing the core Module Suite Content Server modules (Content Script, Beautiful WebForms, Smart Pages) and all the desired Module Suite Extension packages.

  The following screens will guide you through the deployment of Module Suite modules.

- [x] Extract archive:

  Extract ModuleSuite compressed archive file into a temporary location

  ```
  tar -xvzf modulesuite_3_2_0.tar.gz
  ```

- [x] Run installation script and accept EULA:

Run the installation script:

```
./modulsuitesetup.sh
```

and follow the interactive prompts.

Acceptance of the end-user license agreement is mandatory for proceeding with the installation.
A copy of the agreement will be available, upon installation, in:

%OTCS_HOME%/module/amcontentscript_X_Y_Z/license/EULA

Accepting the End User Agreement is mandatory to proceed with the installation.

Enter “Y” when ready.

- Confirm OTCS installation folder:

The installer will prompt you for the location where Content Server is installed. Either confirm (ENTER) the default location or enter the correct location to proceed with the installation.

- Deployment (automatic step):

Automatic import of Content Server dependencies: The installer will automatically attempt to load a few libraries from Content Server.

In case of failure, a warning message could appear during this phase of the installation. In such case, the operation must be performed manually.

- [x] Select extension packages:

Enter “Y” to install the extension when prompted.

Mandatory Components

During the deployment phase, two components are mandatory and MUST be installed:

1. Module Suite Extensions Cache
2. Module Suite Extensions SQL

These components are prerequisites for several Administration tools, including the Content Script Volume Import tool.

Do not skip mandatory components

Failing to install these components may result in certain Administration tools not functioning correctly.

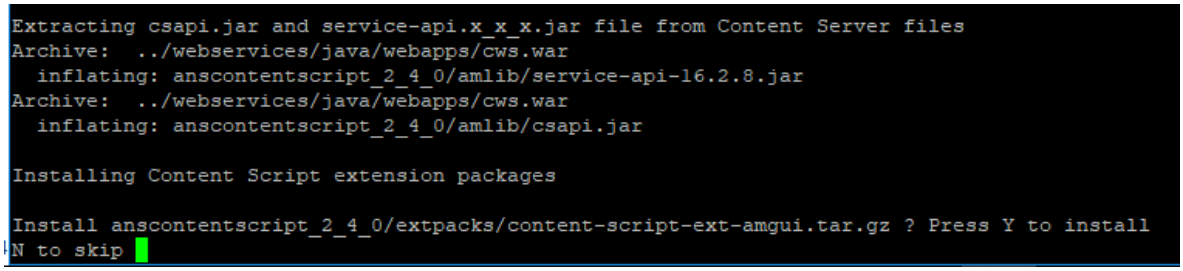
CSEP SAP

The Content Script Extension for SAP™ is a Content Script optional extension package that requires specific additional configuration steps.

It should not be deployed if you are not intending to complete the configuration, as an incomplete configuration could affect the Module Suite functionality.

This extension package requires the SAP™ JCo library *(https://support.sap.com/en/product/connectors/JCo.html)* to be available in the extension repository `<OTHOME>/module/anscontentscript_x_y_z/amlib/sap` and is certified for use with SAP™ JCo version (3.0.6) when used on OpenText Extended ECM and version (3.0.10) when used on CSP. SAP™ JCo library *(https://support.sap.com/en/product/connectors/JCo.html)* can be downloaded from SAP™ website.

More on this extension *(/installation/extpacks/#content-script-extension-for-sap)*.



What to do if the installer raises the error: Unable to automatically extract...

Some Content Script extension packages require two Java libraries that are specific to the target Content Server environment and are not distributed with the module.

The required library files are:

- csapi.jar
- service-api-X.X.XX.jar

and can be found in the web app located in:

```
%OTCS\_HOME%\\webservices\\java\\webapps\\cws.war
```

- classificationsservice-api-X.X.XX.jar

which can be found in the web app located in:

```
%OTCS\_HOME%\\webservices\\java\\webapps\\cs-services-classifications.war
```

- physicalobjectsservice-api-X.X.XX.jar

which can be found in the web app located in:

```
%OTCS\_HOME%\\webservices\\java\\webapps\\cs-services-physicalobjects.war
```

- recordsmanagementservice-api-X.X.XX.jar

which can be found in the web app located in:

```
%OTCS\_HOME%\\webservices\\java\\webapps\\cs-services-recordsmanagement.war
```

To retrieve the files:

- copy the file named XXX.war to a temporary folder
- rename the file XXX.war in XXX.zip
- extract the zip archive contents locate the files in the WEB-INF/lib folder

Once the files have been located, copy them to the folder:

%OTCS_HOME\staging\anscontentscript_x_y_z\amlib

- [x] At this point, the Modules have been deployed in the Content Server Staging folder and is available for installing it through the Content Server administration pages.

Next Steps

Please proceed to the Installation phase.

installation

Module Suite installation guide: Install Modules¶

Overview¶

This guide covers the **Installation** phase that is part of the Module Suite installation guide.

- [x] ~~Deployment~~
- [x] **Installation**
- [ ] Activation
- [ ] Configuration
- [ ] Post-installation patching

This phase covers the Content Server installation of the optional modules previously deployed on the system during the Deployment phase. The related operations will be performed using the Content Server standard administration tools.

Only perform after previous phases are complete

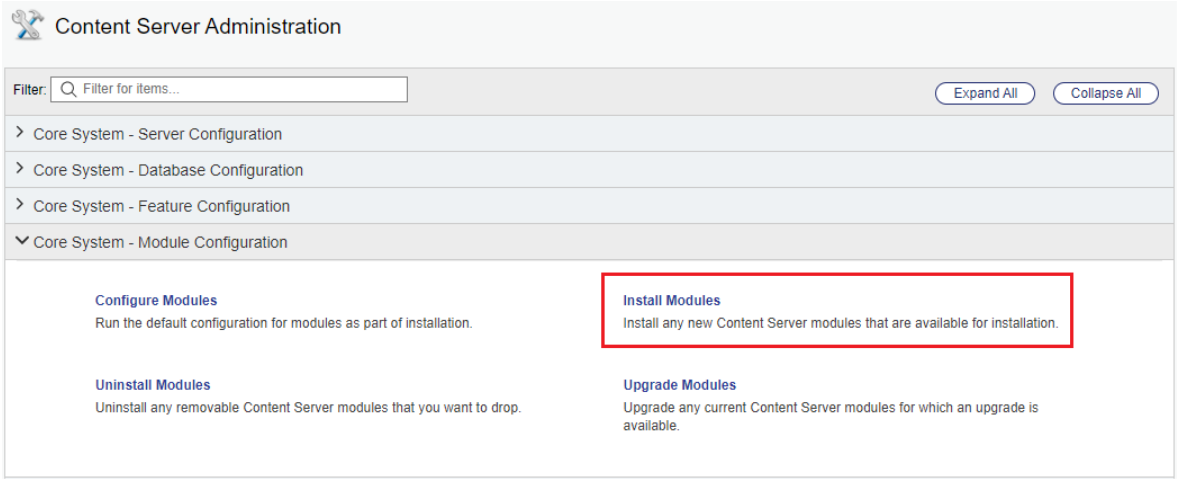
The guide assumes that the Module Deployment phase has already been completed on the target environment. If that is not the case, please go back to the Installation overview.

Step-by-step Installation¶

In order to proceed with the installation of the modules, please follow these steps:

- [x] Start the Content Server services
- [x] Login as Administrator and access the Module administration panel

- [x] Access the Content Server Admin pages > **Core System - Module Configuration** > **Install Modules**



- [x] From the available modules, select **“AnswerModules Content Script x.y.z”**
- [x] Follow the installation steps and restart Content Server when prompted.
- [x] From the Administration Home, access the Module administration panel
- [x] Select **“Install Modules”**
- [x] From the available modules, select **“Answer Modules - Beautiful Web Forms x.y.z”**
- [x] Follow the installation steps and restart Content Server when prompted.
- [x] From the Administration Home, access the Module administration panel
- [x] Select **“Install Modules”**
- [x] From the available modules, select **“Answer Modules - Smart Pages x.y.z”**
- [x] Follow the installation steps and restart Content Server when prompted.
- [x] At this point, the Modules have been installed in the Content Server system.

Apply the available hotfixes¶

- [x] Stop Content Server
- [x] Apply relevant hot fixes
- [x] Start Content Server

Next Steps

- If you plan to apply the license key manually, please proceed to Activation through manual key setup.

- Alternatively, if you plan to import the licensing configuration settings, please proceed to the Activation through key import.

Activate

installation

Module Suite installation guide: Importing the activation key¶

Overview¶

This guide covers the software **Activation** phase that is part of the Module Suite installation guide.

- [x] ~~Deployment~~
- [x] ~~Installation~~
- [x] Activation
- [ ] Configuration
- [ ] Post-installation patching

This phase covers the activation of the modules previously deployed and installed on the system during the Deployment and Installation phases. The related operations will be performed using the Module Suite administration tools, as well as the Content Server standard administration tools.

Only perform after previous phases are complete

The guide assumes that the Module Deployment and Module Installation phases have already been completed on the target environment. If that is not the case, please go back to the Installation overview.

Licensing on a Clustered Environment

Since version 3.1.0, if the installation is performed on a multi-server architecture is no longer necessary to repeat the activation process on all the node of the cluster since the License Key information is stored in the Content Server's database.

Locating the Activation Key in Your Module Suite Fulfillment Document¶

After purchasing Module Suite, you'll receive a fulfillment document containing your activation key. Here's how to find it:

1. Open your fulfillment document.
2. Scroll to the "Software activation" section.
3. Locate the table labeled "Activation key for: Module Suite Named User".
4. Find the row labeled "Activation key".
5. The activation key is the long string of characters in the cell below this label.

Example¶

Software activation

AnswerModules Products activation requires a licensing key. Please cut and paste the following key in your base configuration (more details in the Release Notes).
If no License Key is present, please continue reading below.

Activation key for: Module Suite Named User			
End User			
AnswerModules			
Seats	Fingerprint	Version	Validity
1	**DEMO**	**3.7.0**	**29.08.2024**
Activation key			
rO0ABXfsAGUAbgBkAFUAcwBlAHIAfABBAG4AcwB3AGUAcgBNAG8AZAB1AGwAZQBzAHwAZQB4AHAAaQByAGEAdABpAG8AbgB8ADkAOQA5ADkALQAwADkALQAwADkAfABmAGkAbgBnAGUAcgBwAHIAaQBuAHQAfABUAFIASQBBAEwAfABpAGQAfABBAE0ARQBVAC0AMAAwADAAMAAwADEAfABsAGkAYwBlAG4AcwBlAFQAeQBwAGUAfABQAFIATwBEAFMARAB8AHMAZQBhAHQAcwB8ADEAfAB2AGUAcgBzAGkAbwBuAHwAMwAuADcALgAwAHw=@XUqXR48T4duZRXbAGtm13XA3qFnaTXxPXTVsHRuxR/5dl25vmXn9r60JC1BjhKHL5I3UnyYWJK7kUy3jaZmPFeB59+bLuIEtqpqhF58u5zvDEui37I3PUAj8yV5fJbHqP/ylZt03bzJ4v1vng/mmfCwSi50UlYS5Atb/zlUqkfLjBFF525figkDbxpM9G0eMVXafCcojX58Ik+Tva2WJG4xjLwCe/QXUXXUsQqbTorsjtjp8ADakzlCXUL/LWzT848pRVwRe7/BEu5MkPv68EbpfZej5zJWSxhYNcGFdJUdYoauTE0JG2RCMqSGQzt002xBsK8jz/vBRMp5ypsILfg==			
Module Suite Activation Keys should not include spaces or line feeds. When copying and applying the key, please make sure the key is a single line of text and that no extra characters, including leading and trailing spaces, have been copied with the text.			

Important Note: *License keys are specific to a version of AnswerModules Module Suite . If you are upgrading to a different Module Suite version, you will need a new license key to be generated.*

ModuleSuite activation requires a licensing key. In order to provide the License Key we will need the System Fingerprint of your OTCS instance(s). Please provide the System Fingerprint to our Support team and they will assist with the generation of your keys.
If this is your first setup, we are providing you with a trial license. The sole purpose of this trial license is so that you may setup and start utilizing the product as quickly as possible.
Please provide us with your System Fingerprints so that we may issue your License Key as soon as possible.
When you install the software and/or activate the license(s), you will be agreeing to the terms and conditions of the End User License Agreement included in the product documentation for your software. Please read this agreement carefully prior to activation

Copying the Activation Key

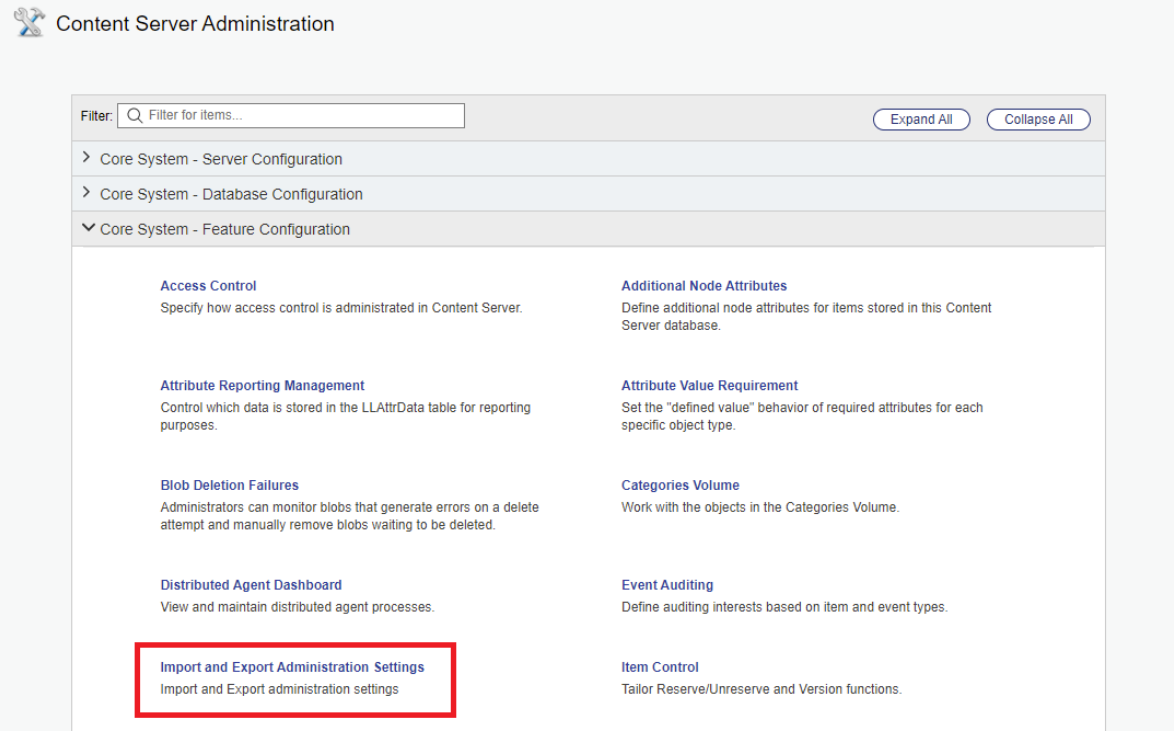
- Copy the entire key as a single line of text.
- Do not include any extra characters, leading/trailing spaces, or line feeds.
- Double-check for accuracy when copying.

Version Specificity

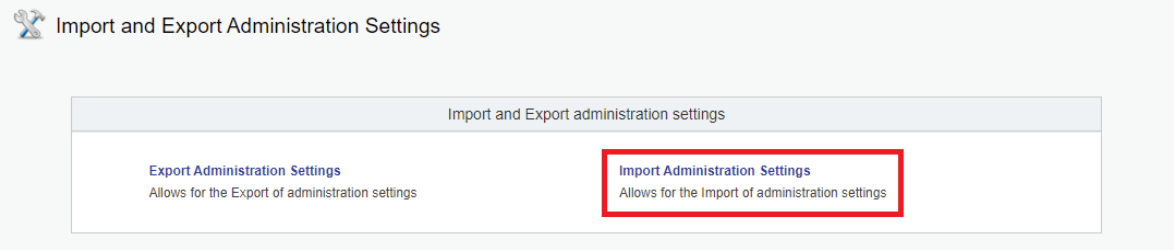
The activation key is specific to the Module Suite version listed in the document. For upgrades to different versions, you'll need a new license key generated.

Importing the License Key¶

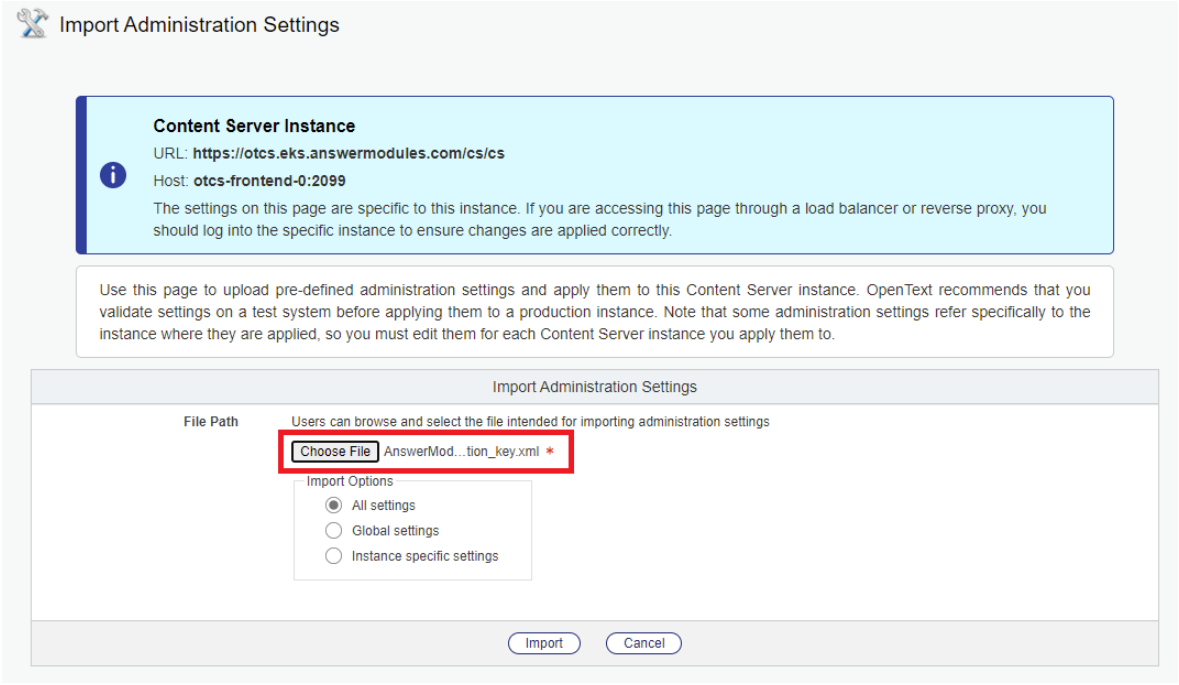
- [x] As the system Admin user, open the Content Server Administration pages.
- [x] Locate the **Core System - Feature Configuration** section. Within this section, open the **Import and Export Administration Settings** tool.



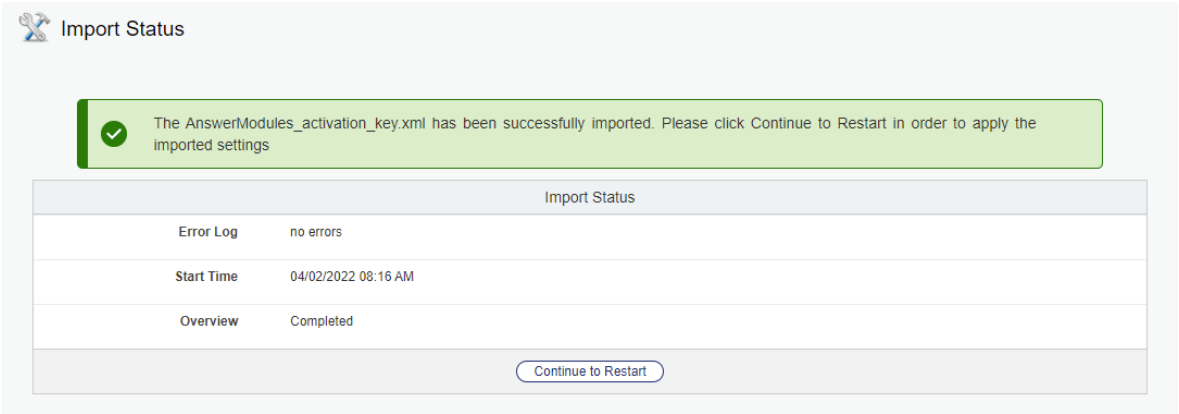
- [x] Within the Import and Export Administration Settings page, locate the **Import Administration Settings** entry.



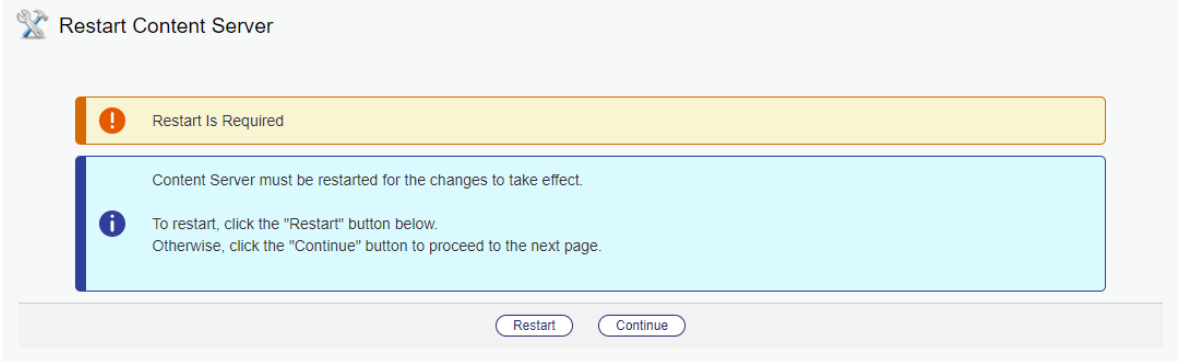
- In the **File Path** field, locate and select the AnswerModules Activation Key XML file. Then, click on "Import".



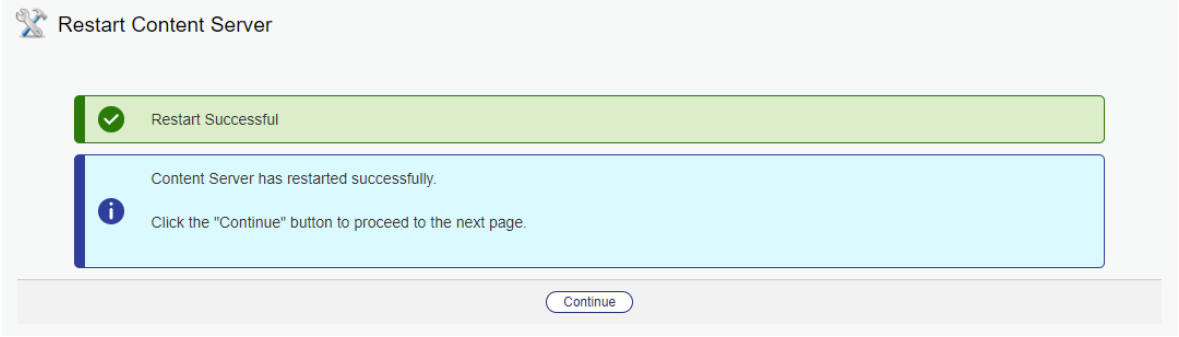
- Updating the activation key requires a system restart. Click "Continue to Restart" to be redirected to the **Restart** page.



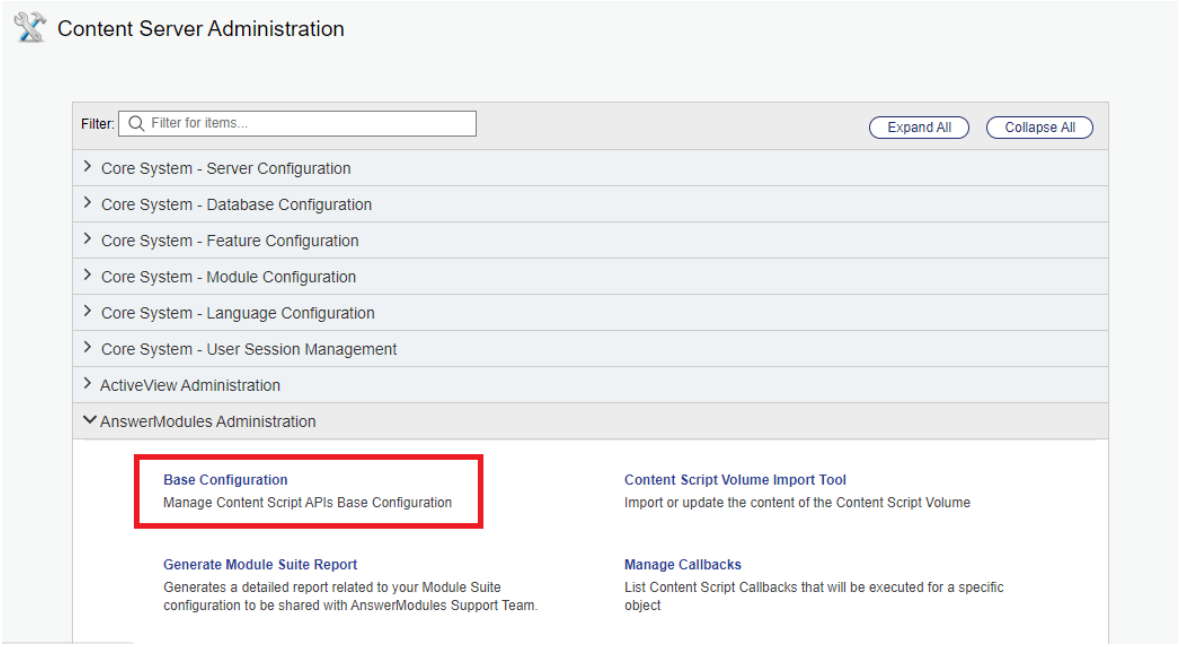
- Click **Restart**



- [x] Wait for the system to complete the Restart operation. Once complete, click **Continue**.

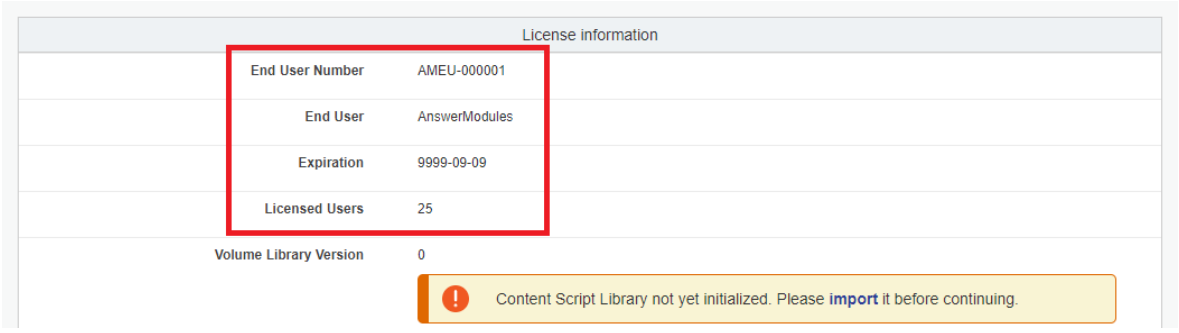


- [x] After the restart, navigate to the AnswerModules administration pages to check the results of the activation operations.
  In the Content Server Administration pages, locate the **AnswerModules Administration** section. Within this section, open the **Base Configuration** tool.



- [x] At the top of the Base Configuration page, check the validity of the newly applied key.

  When a valid activation key is present, the key details will be visibile to the administrator.



Next Steps

Please proceed to the Configuration phase.

installation

Module Suite installation guide: Manually setting the activation key¶

Overview¶

This guide covers the software **Activation** phase that is part of the Module Suite installation guide.

- [x] ~~Deployment~~
- [x] ~~Installation~~
- [x] Activation
- [ ] Configuration
- [ ] Post-installation patching

This phase covers the activation of the modules previously deployed and installed on the system during the Deployment and Installation phases. The related operations will be performed using the Module Suite administration tools, as well as the Content Server standard administration tools.

Only perform after previous phases are complete

The guide assumes that the Module Deployment and Module Installation phases have already been completed on the target environment. If that is not the case, please go back to the Installation overview.

Licensing on a Clustered Environment

Since version 3.1.0, if the installation is performed on a multi-server architecture is no longer necessary to repeat the activation process on all the node of the cluster since the License Key information is stored in the Content Server's database.

Locating the Activation Key in Your Module Suite Fulfillment Document¶

After purchasing Module Suite, you'll receive a fulfillment document containing your activation key. Here's how to find it:

1. Open your fulfillment document.

2. Scroll to the "Software activation" section.
3. Locate the table labeled "Activation key for: Module Suite Named User".
4. Find the row labeled "Activation key".
5. The activation key is the long string of characters in the cell below this label.

Example¶

Software activation

AnswerModules Products activation requires a licensing key. Please cut and paste the following key in your base configuration (more details in the Release Notes).
If no License Key is present, please continue reading below.

Activation key for: Module Suite Named User			
End User			
AnswerModules			
Seats	Fingerprint	Version	Validity
1	**DEMO**	**3.7.0**	**29.08.2024**
Activation key			
rO0ABXfsAGUAbgBkAFUAcwBlAHIAfABBAG4AcwB3AGUAcgBNAG8AZAB1AGwAZQBzAHwAZQB4AHAAaQByAGEAdABpAG8AbgB8ADkAOQA5ADkALQAwADkALQAwADkAfABmAGkAbgBnAGUAcgBwAHIAaQBuAHQAfABUAFIASQBBAEwAfABpAGQAfABBAE0ARQBVAC0AMAAwADAAMAAwADEAfABsAGkAYwBlAG4AcwBlAFQAeQBwAGUAfABQAFIATwBEAFMARAB8AHMAZQBhAHQAcwB8ADEAfAB2AGUAcgBzAGkAbwBuAHwAMwAuADcALgAwAHw=@XUqXR48T4duZRXbAGtm13XA3qFnaTXxPXTVsHRuxR/5dl25vmXn9r60JC1BjhKHL5I3UnyYWJK7kUy3jaZmPFeB59+bLuIEtqpqhF58u5zvDEui37I3PUAj8yV5fJbHqP/ylZt03bzJ4v1vng/mmfCwSi50UlYS5Atb/zlUqkfLjBFF525figkDbxpM9G0eMVXafCcojX58Ik+Tva2WJG4xjLwCe/QXUXXUsQqbTorsjtjp8ADakzlCXUL/LWzT848pRVwRe7/BEu5MkPv68EbpfZej5zJWSxhYNcGFdJUdYoauTE0JG2RCMqSGQzt002xBsK8jz/vBRMp5ypsILfg==			
Module Suite Activation Keys should not include spaces or line feeds. When copying and applying the key, please make sure the key is a single line of text and that no extra characters, including leading and trailing spaces, have been copied with the text.			

Important Note: *License keys are specific to a version of AnswerModules Module Suite . If you are upgrading to a different Module Suite version, you will need a new license key to be generated.*

ModuleSuite activation requires a licensing key. In order to provide the License Key we will need the System Fingerprint of your OTCS instance(s). Please provide the System Fingerprint to our Support team and they will assist with the generation of your keys.
If this is your first setup, we are providing you with a trial license. The sole purpose of this trial license is so that you may setup and start utilizing the product as quickly as possible.
Please provide us with your System Fingerprints so that we may issue your License Key as soon as possible.
When you install the software and/or activate the license(s), you will be agreeing to the terms and conditions of the End User License Agreement included in the product documentation for your software. Please read this agreement carefully prior to activation

Copying the Activation Key

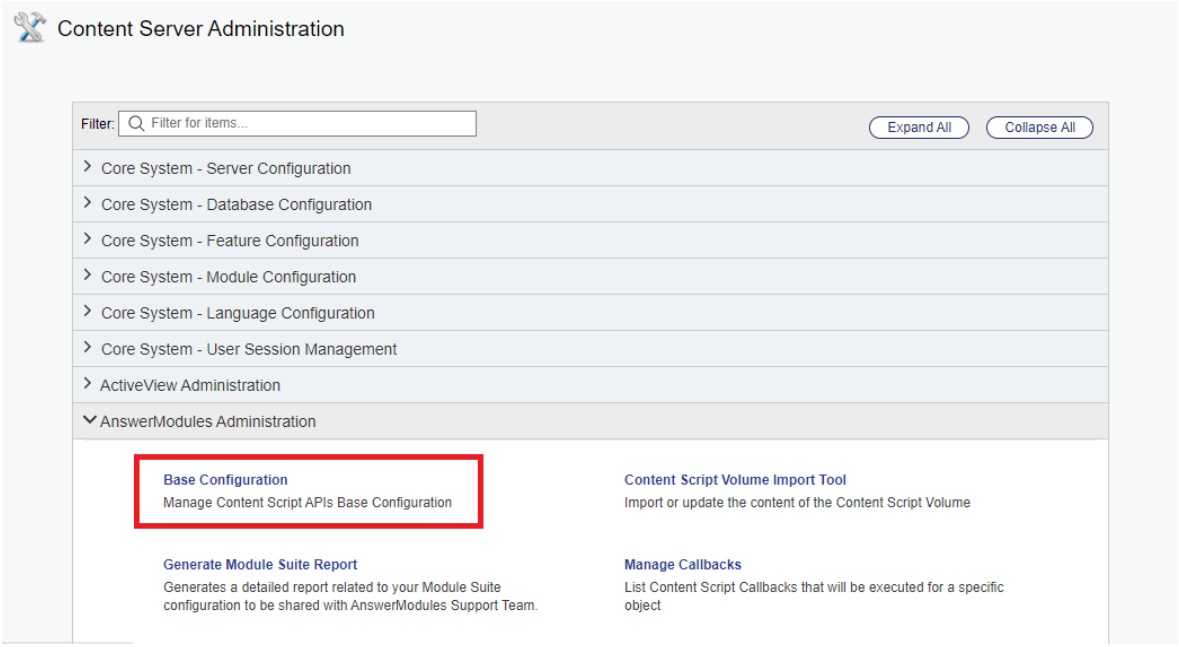
- Copy the entire key as a single line of text.
- Do not include any extra characters, leading/trailing spaces, or line feeds.
- Double-check for accuracy when copying.

Version Specificity

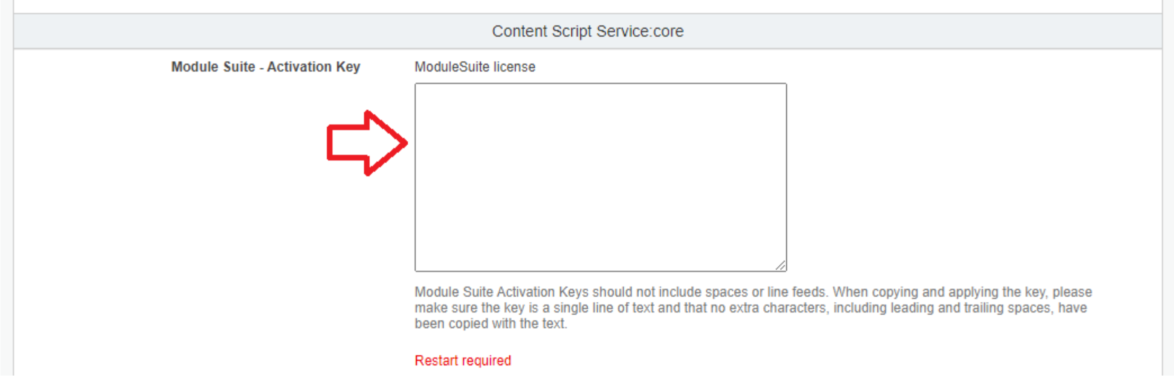
The activation key is specific to the Module Suite version listed in the document. For upgrades to different versions, you'll need a new license key generated.

Applying the License Key manually¶

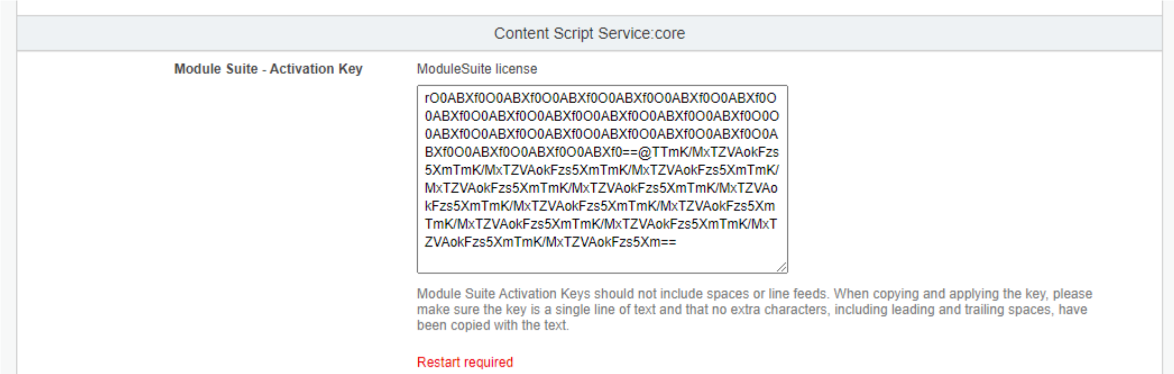
- [x] As the system Admin user, open the Content Server Administration pages.
- [x] Locate the **AnswerModules Administration** section. Within this section, open the **Base Configuration** tool.



- [x] Within the Base Configuration page, locate the **Module Suite - Activation Key** entry (it can be found in the **Core** section).



- Enter the activation key in the text area.

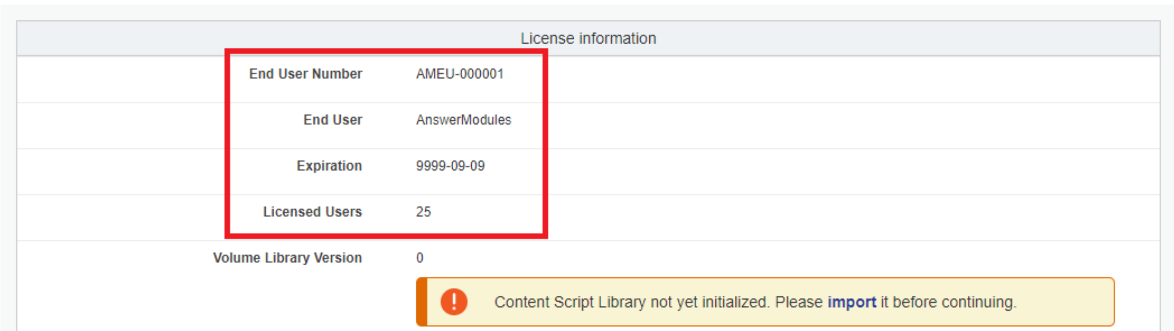


License key format

When copying the license key, make sure that no whitespaces or new line characters are included.

- Save the Base Configuration and restart Content Server when prompted.
- After the restart, check the validity of the newly applied license at the top of the **Base Configuration** page.

When a valid license is present, the license details will be visibile to the administrator.



Next Steps

Please proceed to the Configuration phase.

installation

Module Suite installation guide: Initial Configuration¶

Overview¶

This guide covers the **Configuration** phase that is part of the Module Suite installation guide.

- [x] ~~Deployment~~
- [x] ~~Installation~~
- [x] ~~Activation~~
- [x] Configuration
- [ ] Post-installation patching

This phase covers the minimal configuration of the optional modules previously deployed and installed on the system during the Deployment and Installation phases. The related operations will be performed using the Module Suite administration tools.

Only perform after previous phases are complete

The guide assumes that the Module Deployment, Installation and Activation phases have already been completed on the target environment. If that is not the case, please go back to the Installation overview.

Importing the core library components¶

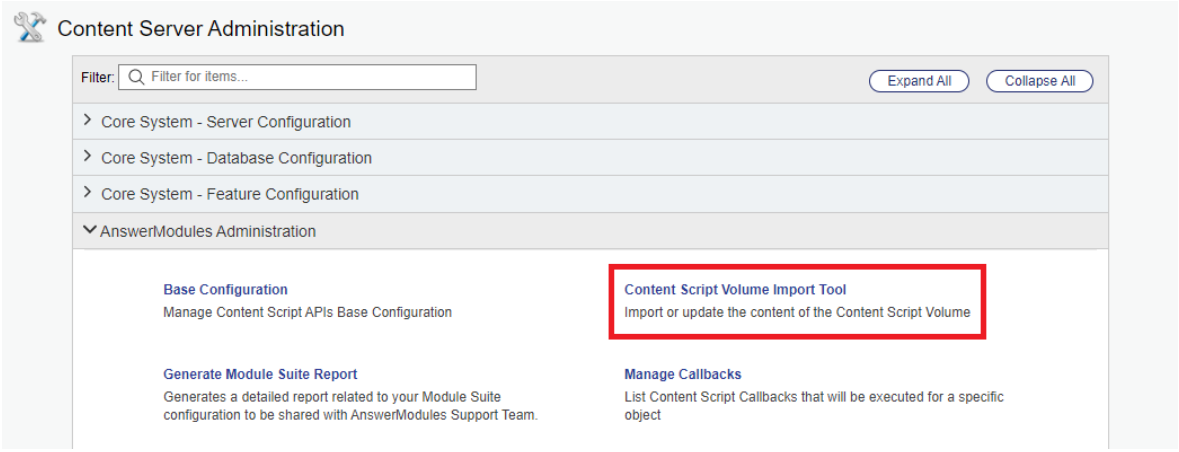
Once the software has been activated, it is possible to proceed with the setup of the minimal configuration settings required to run Module Suite. This includes creating a number of runtime elements within the "Content Script Volume", a special container for Module Suite configuration objects.

More details on the Content Script Volume and its structure can be found here.

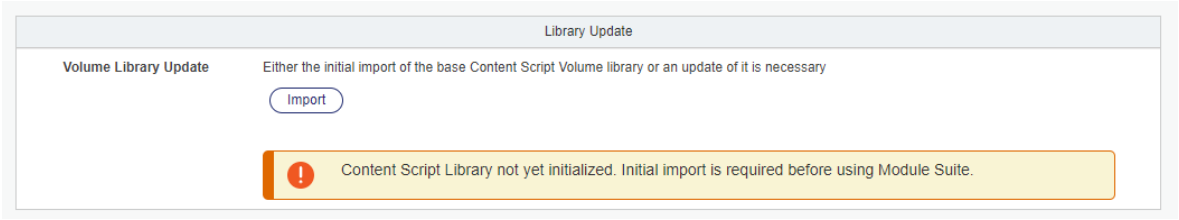
Proceed with the following steps to complete the initial configuration.

- [x] As the system Admin user, open the Content Server Administration pages.

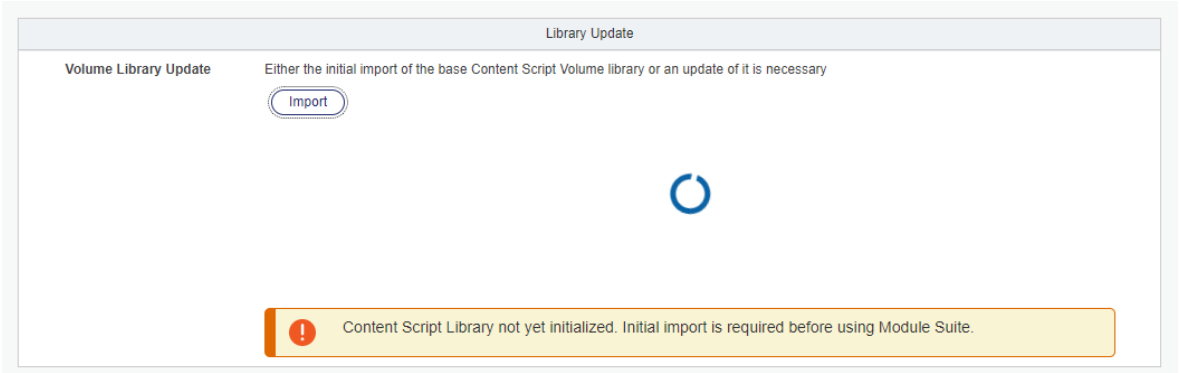
- [x] Locate the **AnswerModules Administration** section. Within this section, open the **Content Script Volume Import** tool.



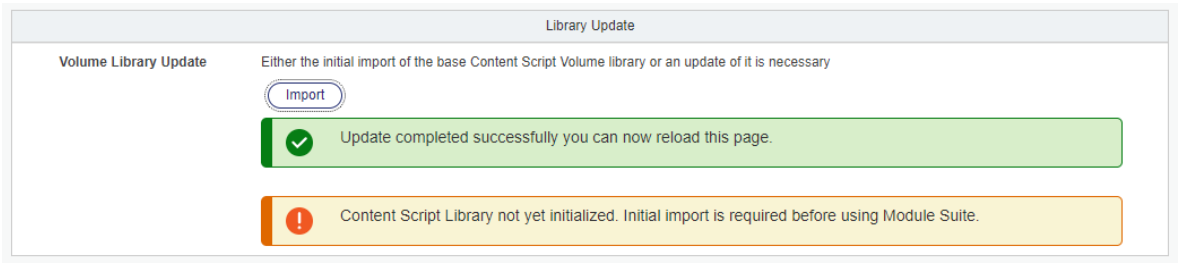
- [x] Within the Content Script Volume Import page, locate the **Library Update** section (it can be found at the very top of the page).
  Click the **Import** button.



- [x] Wait for the import operation to complete.

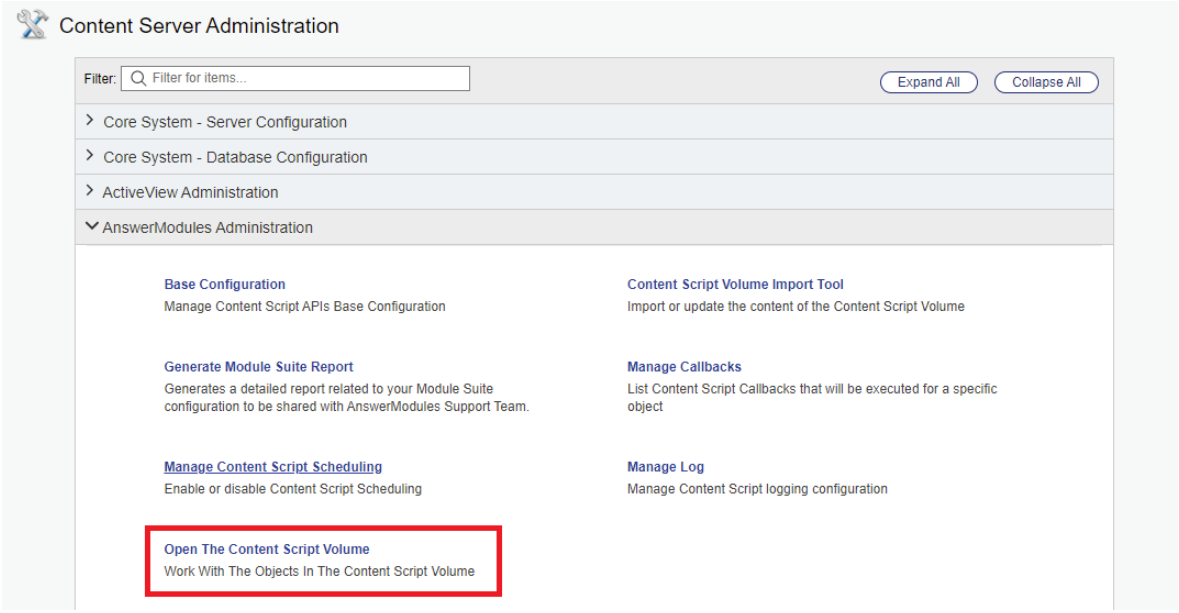


- [x] Once the operation is complete, a success message will be shown. Upon refreshing the page, the **Library Update** section will no longer be shown.

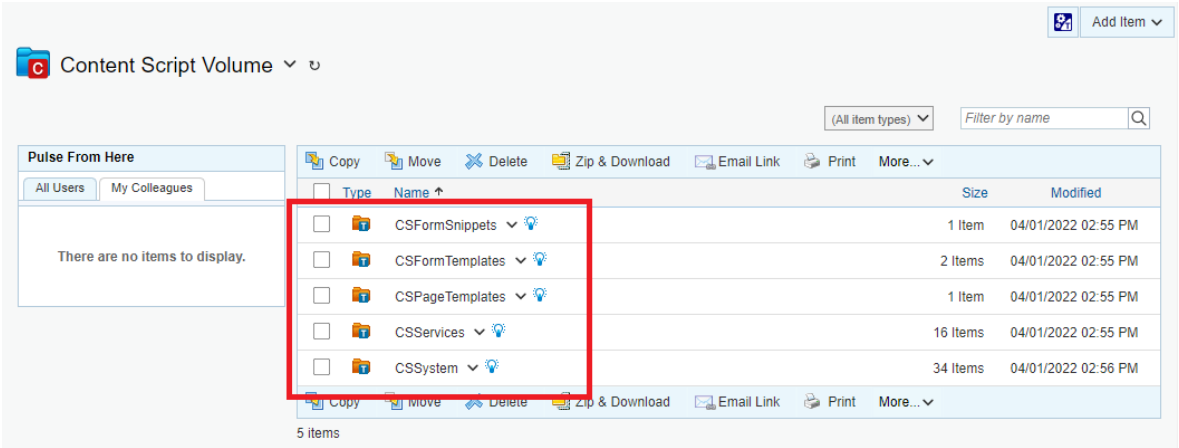


- [x] After the restart, navigate to the AnswerModules administration pages to check the results of the import operations.

In the Content Server Administration pages, locate the **AnswerModules Administration** section. Within this section, click on the **Open the Content Script Volume** link.



☑ The following minimum set of folders will have been created in the Content Script Volume.



Next Steps

Please proceed to the Post installation patching phase.

installation

Module Suite installation guide: Install Hotfixes¶

Overview¶

This guide covers the **Post-installation patching** phase that is part of the Module Suite installation guide.

- ~~Deployment~~
- ~~Installation~~
- ~~Activation~~
- ~~Configuration~~
- Post-installation patching

This phase refers to the final operations that are required to ensure that the target system is up to date with all relevant software patches and hotfixes.

Only perform after previous phases are complete

The guide assumes that the Module **Deployment**, **Installation**, **Activation** and **Configuration** phases have already been completed on the target environment. If that is not the case, please go back to the Installation overview.

Applying patches¶

Each Module Suite patch is released with its own Patch Notes and (optionally) with specific installation tasks. Please refer to the generic Applying Hotfixes guide for detailed information on this topic.

Installation complete

Congratulations! The Module Suite's initial setup is now complete.

clustered installation installation

Installing Module Suite on a clustered environment¶

In a Content Server cluster environment, it is mandatory to install Module Suite modules on each node that makes up the cluster.

The installation process in a cluster is more complex than installing on a single server, as a slightly different procedure must be performed on each remaining node in the cluster after installing the modules on the first one. The recommended approach is to install Module Suite on a **primary node** (the node on which the primary OpenText Admin Content Server services are installed and configured) and then copy the installed modules to each node in the cluster. This approach ensures that all installed modules are identical and that the patch level on all nodes is the same.

> We will refer to the Content Server installation directory as **%OTCS_HOME%**.

Deployment on the primary node¶

Module Suite package installation on a Primary node is identical to the installatoin process into the non-clustered environment.

- ☑ Stop Content Server services on all the nodes in the cluster
- ☑ Proceed with the Module Suite installation on the **Primary node**

  Detailed description of this procedure can be found in Installing Module Suite guide.

Deployment on the secondary node(s)¶

Once the Module Suite modules are installed on the primary node, the module packages can be deployed on the remaining cluster nodes.
Proceed with the following installation steps on all **Secondary nodes**

- ☑ Make a copy of the following resources and make them available in a working folder on the Secondary node:

  1. %OTCS_HOME%/module/anscontentscript_x_x_x
  2. %OTCS_HOME%/module/ansbwebform_x_x_x
  3. %OTCS_HOME%/module/anscontentsmartui_x_x_x
  4. %OTCS_HOME%/support/anscontentscript
  5. %OTCS_HOME%/support/ansbwebform
  6. %OTCS_HOME%/support/anscontentsmartui

  > **Installed Modules**
  >
  > The actual folders to be copied depend on the Module that have been installed on the Primary node.
  > e.g. if you are only installing Content Script and Beautiful WebForms modules, the "anscontensmartui" folders will not be available in the Primary node.

- ☑ On the Secondary node, ensure that all Content Server services are stopped.

- [x] On the Secondary node, copy all the modules and support folders previously identified to their corresponding target location within %OTCS_HOME%.
- [x] On the Secondary node, proceed to manually **reconcile the opentext.ini** file in **%OTCS_HOME%/config**.
  Pay particular attention to the [Modules] and [javaserver] sections on the opentext.ini file.
- [x] On the Secondary node, start the Content Server services.

container installation

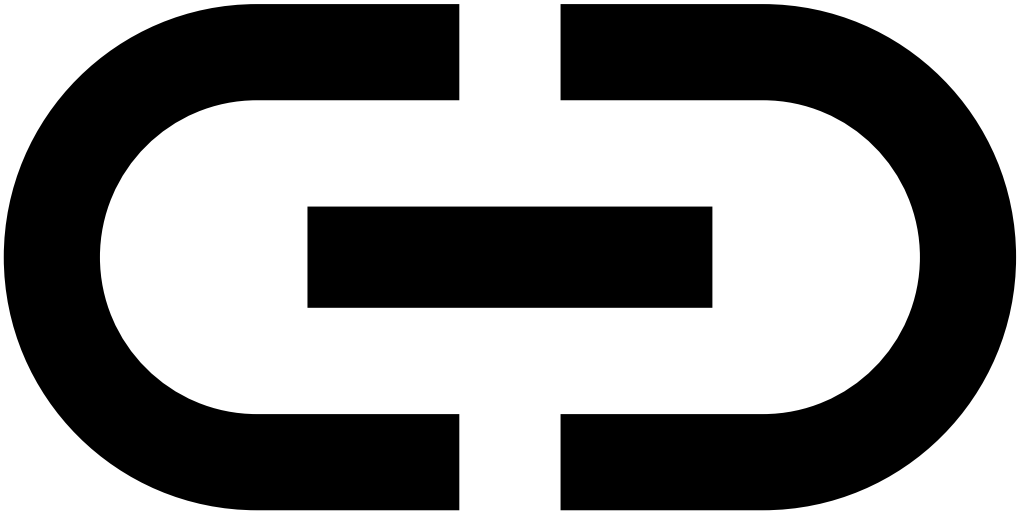
Install Module Suite on OpenText Extended ECM CE¶

Overview of the installation phases¶

This guide provides the complementary steps of the **OpenText Extended ECM CE 2X.Y - Cloud Deployment Guide** for deploying the Module Suite on the OpenText Extended ECM 2X.Y in a Kubernetes Cluster.

Check the official OpenText documentation

This guide is based on the OpenText Extended ECM CE 24.2 - Cloud Deployment Guide



(https://webapp.opentext.com/piroot/sulccd/v240300/sulccd-igd/en/html/_manual.htm). Procedures may vary for other versions of OpenText Extended ECM. Always refer to the appropriate "OpenText Extended ECM CE X.Y - Cloud Deployment Guide" for your specific version, as parameters and arguments discussed in this manual may differ.

What is covered by this guide¶

This guide covers the following high-level phases:

1. **Build Init Containers:** This phase covers the deployment of the Module Suite software binaries on an external image repository in the form of **Init containers**. The operations are performed using command-line commands.

2. **Deployment:** This phase covers the definition of the arguments to be added to the standard Helm installation command for deploying the Module Suite and OpenText

Extended ECM in the target Kubernetes Cluster. The operations are performed using command-line commands.

Activation and Configuration

For detailed instructions on activation and configuration, always consult the most recent version of the official Module Suite documentation. For information on activating the Module Suite software, please refer to the official documentation: Activate and Import Module Suite *(https://developer.answermodules.com/manuals/current/installation/modulesuite/activate_import/)* For information on post-installation configuration steps, including importing core libraries and components, please refer to the official documentation: Configure Module Suite *(https://developer.answermodules.com/manuals/current/installation/modulesuite/configure/)*

Prerequisites¶

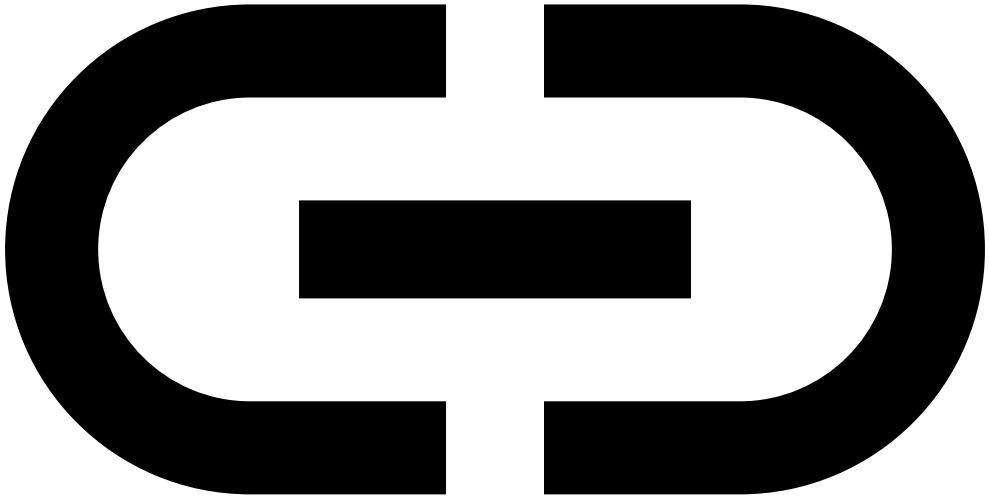
A host PC or VM is required for leveraging the deployment/installation process. This host should contain the following software:

- The destination Kubernetes cluster hyperscaler client software
- Docker
- Kubectl
- Helm
- The latest publicly available Alpine Linux image

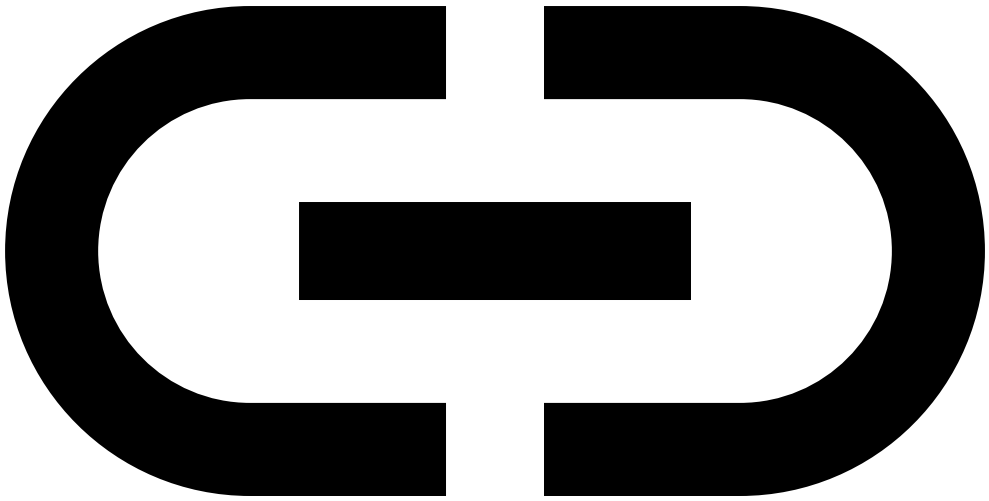
This guide presumes the usage of a Linux host PC or VM.

Software Download References¶

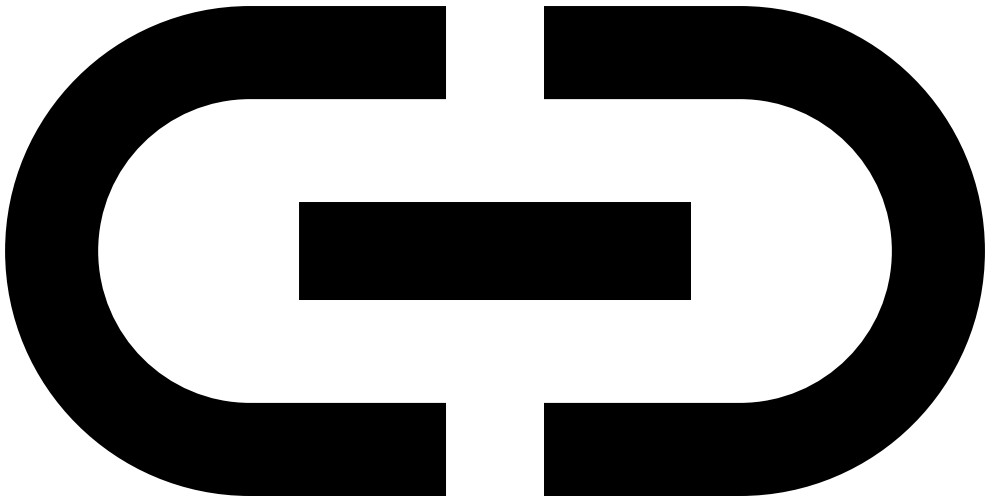
1. **Kubernetes cluster hyperscaler client software**: AWS: AWS CLI



(https://aws.amazon.com/cli/), Azure: Azure CLI

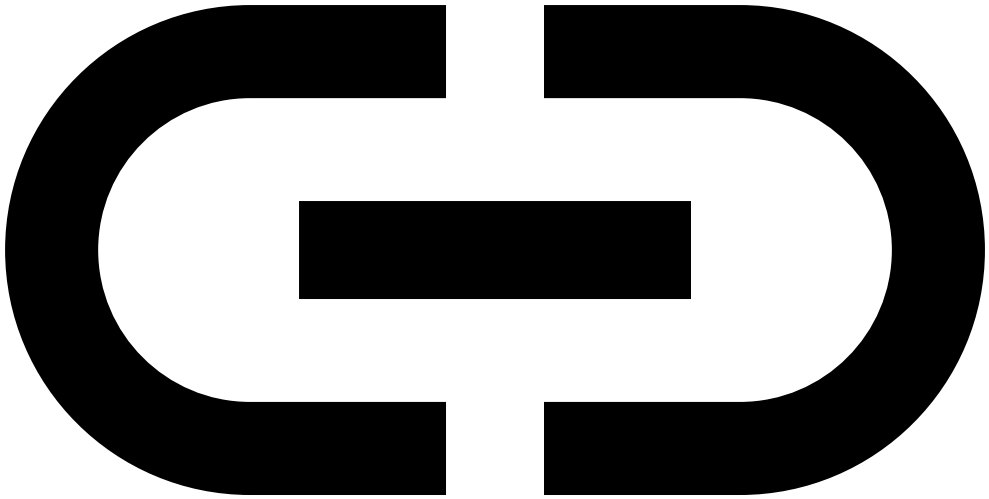


(https://docs.microsoft.com/en-us/cli/azure/install-azure-cli), Google Cloud: Google Cloud SDK



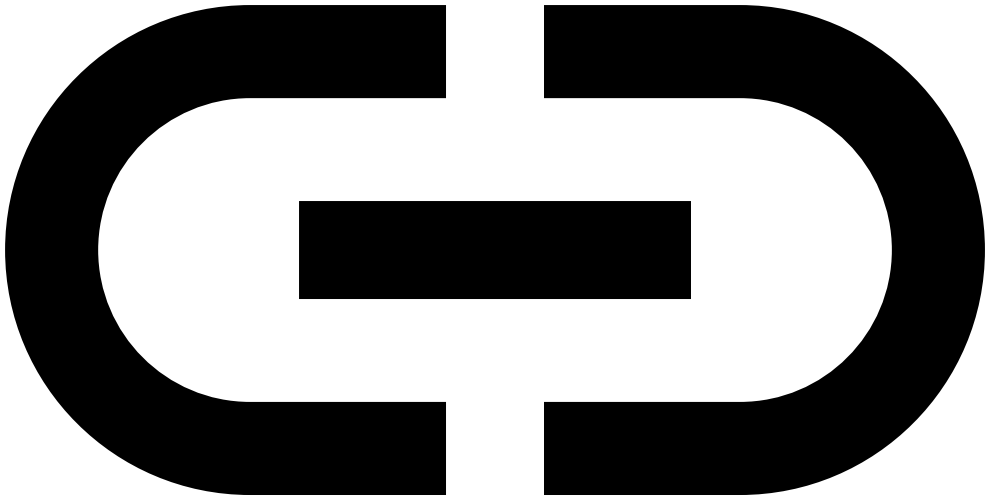
(https://cloud.google.com/sdk/docs/install)

2. **Docker**: Docker Engine installation



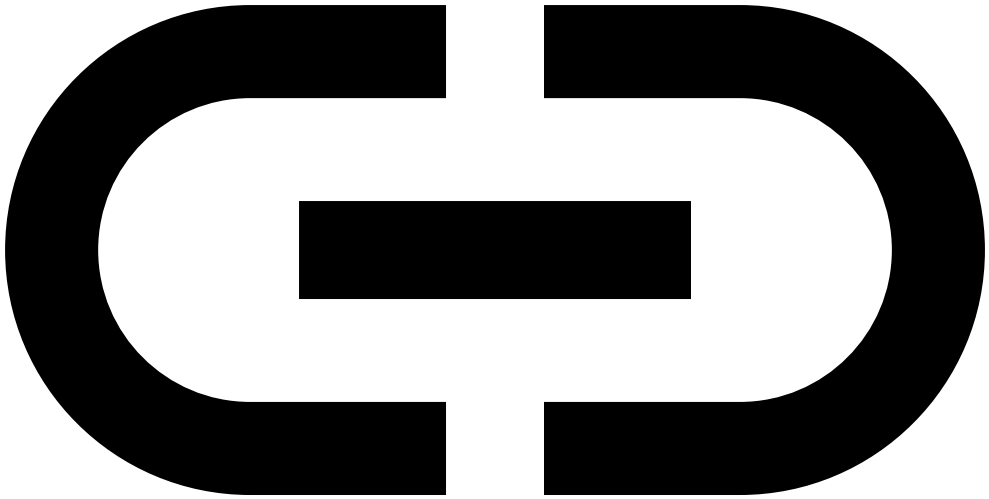
(https://docs.docker.com/engine/install/)

3. **Kubectl**: kubectl installation



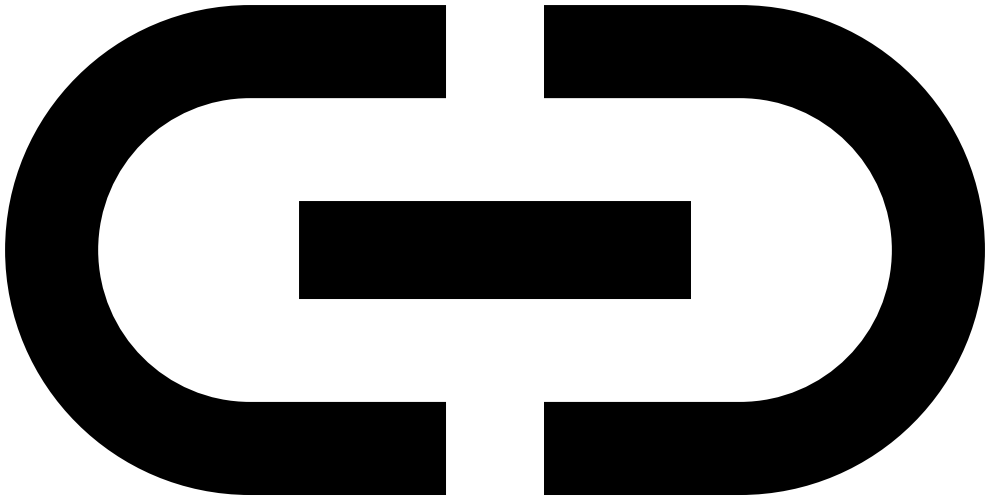
(https://kubernetes.io/docs/tasks/tools/)

4. **Helm**: Helm installation



(https://helm.sh/docs/intro/install/)

5. **Alpine Linux image**: Alpine Linux Docker image



(https://hub.docker.com/_/alpine)

Additional Requirements¶

1. **Module Suite System Center compatible artifacts**: Module Suite modules suitable for installation with OpenText System Center.

   Check Module Suite release notes for compatibility

   Always verify on the Module Suite release notes the compatibility of the Module Suite version with the OpenText Extended ECM CE application.

2. **AnswerModules activation key**: Either in plain text format or in an OTCS Configuration Export XML format. The XML format is recommended to prevent errors due to manual input.

How to get System Center artifacts and Activation Key

To obtain the Module Suite artifacts and its activation key, open a request to the AnswerModules Support Team at Request Activation Key *(mailto:support@answermodules.zohosupport.com?subject=Activation%20Key%20Request&body=We%20are%20requesting%20a%20valid%20activation%20key%20*

3. **External image repository**: Available for hosting the AnswerModules Module Suite init containers.

This guide presumes the usage of Dockerhub as the external image repository.

Please ensure you download and install the appropriate versions compatible with your system and the version of OpenText Extended ECM CE you are deploying.

Build Init Containers¶

In this phase, we'll prepare the AnswerModules Module Suite Init containers and upload them to an external container repository. This crucial step enables the integration of Module Suite with OpenText Extended ECM in a Kubernetes environment.

Understanding Init Containers¶

Init containers are a powerful feature in Kubernetes that run before the main application containers in a pod. They serve several important purposes:

1. **Environment preparation**: Set up necessary configurations or data.
2. **Dependency checks**: Ensure required services are available.
3. **Initialization tasks**: Perform one-time setup operations.

In our specific use case, init containers will add the Module Suite modules to the OpenText Extended ECM installation, ensuring all necessary components are in place before the main application starts.

More about Init Containers

For more detailed information about init containers, refer to the official Kubernetes documentation: Init Containers | Kubernetes *(https://kubernetes.io/docs/concepts/workloads/pods/init-containers/)*

Get all you need

Before proceeding, please ensure you have:

1. A Dockerhub account (or access to another container registry).
2. Familiarity with:
    - Unix systems and commands

- Docker and its command-line interface
- Basic Kubernetes concepts

Step-by-Step Procedure¶

Access the Linux host PC or VM designated for the installation process and follow these steps:

Init Container FS structure¶

1. **Create main folder**
   - Create a new folder on the host machine (e.g., `MSInitContainer`)
2. **Create subfolder**
   - Inside the main folder, create a subfolder (e.g., `Init`)
3. **Add Dockerfile**
   - Copy the `Dockerfile_init` file (obtained from OpenText My Support) into the `Init` folder
4. **Create the extensions folder**
   - Inside the `Init` folder, create a new folder named `extensions`
5. **Set up the extensions folder structure**
   - Navigate to the `extensions` folder
   - Create the following folder structure:

```
extensions/
├── apps/
├── language_packs/
├── manifest/
├── module_language_packs/
├── modules/
└── patch/
```

6. **Set up apps folder structure**
   - Navigate to the `extensions/apps/` folder
   - Create the following folder structure:

```
apps/
├── install/
└── upgrade/
```

Check your work

Your final folder structure should look like this:

```
MSInitContainer/
└── Init/
├── Dockerfile_init
└── extensions/
├── apps/
│   ├── install/
│   └── upgrade/
├── language_packs/
├── manifest/
├── module_language_packs/
├── modules/
└── patch/
```

1. Copy Module Suite artifacts
   - Copy the first module suite artifact **anscontentscript_X_Y_Z_GA.tar.gz** in the **modules** folder. Do **NOT** expand the artifact archives.

Only one artifact at a time

Important do not add more than a module at a time because we are going to build separate Init Containers for each one of them

Build the container¶

1. Open a command shell, navigate to the main folder (i.e. `MSInitContainer`) and build the Init Containers by executing commands below:

```
docker build -f Dockerfile_init . --build-arg base_image_tag=a.b.c --tag anscontentscript:x.y
```

```
docker build -f Dockerfile_init . --build-arg base_image_tag=a.b.c --tag anscontentsmartui:x.
```

```
docker build -f Dockerfile_init . --build-arg base_image_tag=a.b.c --tag ansbwebform:x.y.z.t
```

where **a.b.c** is the version of the Alpine Linux image, **module** is the name of the artifact for which you are creating the init container, **x.y.z** is the version of the Module Suite module and **t** is the version of the Init container. E.g.

```
docker build -f Dockerfile_init . --build-arg base_image_tag=3.6.0 --tag anscontentscript:3.7
```

Follow the building process execution

You can follow the building process using the command shell, where messages are constantly updated. if terminated successfully you should readm a message similar to the one below

```
Successfully built <Image_ID>
Successfully tagged <Image_Name>:<Tag_Name>
```

Push the Init Containers to your image repository¶

Once built, the init container you can push them to your image repository, assuming you are using Dockerhub you can proceed as follows:

1. Tag the init container with the Dockerhub account, owner, of the repository:

```
docker tag anscontentscript:x.y.z.t dockerhubuser/anscontentscript:x.y.z.t
docker tag ansbwebform:x.y.z.t dockerhubuser/ansbwebform:x.y.z.t
docker tag anscontentsmartui:x.y.z.t dockerhubuser/anscontentsmartui:x.y.z.t
```

e.g.

```
docker tag anscontentscript:3.7.0.1 dockerhubuser/anscontentscript:3.7.0.1
docker tag ansbwebform:3.7.0.1  dockerhubuser/ansbwebform:3.7.0.1
docker tag anscontentsmartui:3.7.0.1 dockerhubuser/anscontentsmartui:3.7.0.1
```

1. Push the init container to the external image repository as follows:

```
docker push dockerhubuser/anscontentscript:x.y.z.t
docker push dockerhubuser/ansbwebform:x.y.z.t
docker push dockerhubuser/anscontentsmartui:x.y.z.t
```

e.g.

```
docker push dockerhubuser/anscontentscript:3.7.0.1
docker push dockerhubuser/ansbwebform:3.7.0.1
docker push dockerhubuser/anscontentsmartui:3.7.0.1
```

Follow the process execution

You can follow the process using the command shell, where messages are constantly updated. If terminated successfully you should readm a message similar to the one below

```
_The push refers to repository \[_[_docker.io/dockeruser/anscontentscript_](http://docker.io/msd(

_e6bed7bffb32: Pushed_

_8be46d384520: Pushed_

_24302eb7d908: Pushed_
```

```
_3.7.0.1: digest: sha256:33dc6c3810b0e5a72cfa7fc98fd1f4780fe3aaac320bc715c8a4233 size: 949_
```

Deploy¶

Enable extensions in Helm deployment¶

- When deploying OpenText Extended ECM, include this Helm command argument:

```
--set otcs.config.extensions.enabled=true
```

Specify Init container details¶

- For each Init container (including any manifest container), add the following Helm command arguments:

```
--set otcs.initContainers[n].name=<Init_Container_Image_Name>
--set otcs.initContainers[n].image.source=<Image_Source>
--set otcs.initContainers[n].image.name=<Image_Name>
--set otcs.initContainers[n].image.tag=<Image_Tag>
```

Replace `[n]` with an incrementing number for each Init container, starting from 0.

Parameter Details

- `<Init_Container_Image_Name>`: A name of your choice for the Init container.
- `<Image_Source>`: The registry containing your images. Include this even if it's the same as in the `<platform>.yaml` file.
- `<Image_Name>`: The name you set in your docker build command.
- `<Image_Tag>`: The tag you set in your docker build command.

E.g.

```
helm install myotxecm otxecm -f otxecm/platforms/gcp.yaml
--set otds.otdsws.cryptKey=MTIzNDU2Nzg5YWNiZGVmZw==
--set otcs.config.extensions.enabled=true
--set otcs.initContainers[0].name=anscontentscript
--set otcs.initContainers[0].image.source=docker.io
--set otcs.initContainers[0].image.name=dockerUser/anscontentscript
--set otcs.initContainers[0].image.tag=3.7.0.1
--set otcs.initContainers[1].name=ansbwebform
--set otcs.initContainers[1].image.source=docker.io
--set otcs.initContainers[1].image.name=dockerUser/ansbwebform
--set otcs.initContainers[1].image.tag=3.7.0.1
--set otcs.initContainers[2].name=anssmartui
--set otcs.initContainers[2].image.source=docker.io
--set otcs.initContainers[2].image.name=dockerUser/anssmartui
--set otcs.initContainers[2].image.tag=3.7.0.1
```

(Optional) Activate the Module Suite

The activation of the Module Suite can be performed when the containers are deployed.

1. Add the administration settings file to the Helm chart Place an Administration Settings file named adminSettings.xml (containing the ModuleSuite activation key) in the ../otxecm/charts/otcs folder.
2. Enable the use of an administration settings file Include the Helm command argument

```
--set otcs.loadAdminSettings.enabled=true
```

Upgrading Module Suite

upgrade

Getting ready to upgrade Module Suite¶

Whenever a new release of Module Suite is released, it is highly recommended for customers to update their installation. New releases not only contains fixes for the identified bugs, but most importantly new features that might open new usage scenarios for your Module Suite applications. Updating Module Suite is quite a straight forward procedure, that should take between 15 to 45 minutes (depending on how complex your Content Server architecture is). The system down time is limited to the two restarts required for each node.

Overview of the Module Suite upgrade process¶

This guide describes the step-by-step procedure that will lead to upgrading your Module Suite installation on a Content Server environment.

The upgrade procedure reflects most of the same steps that are performed upon initial installation.

Depending on the characteristics of the target environment (Unix/Linux or Windows, single server or clustered, ...) different options might be provided for each installation phase.

The following high-level phases will be covered:

1. Deployment
   This phase covers the deployment of the software binaries on the target system. The related operations will be typically performed with a click-through installer.

2. Module Upgrade
   This phase covers the "upgrade" phase of the updated Modules within the target Content Server system. The operation is performed through the standard OpenText Content Server Administration tools.

3. Activation
   This phase covers the available procedures to apply the required software keys and activate the Module Suite software. The operations are performed using AnswerModules Administration tools available within the Content Server Admin pages and standard OpenText Content Server Administration tools.

4. Configuration

   This phase covers the minimum set of post-installation configuration steps that are necessary to get the software up and running. This includes importing or updating certain core libraries and components in the system, as well as resolving conflicts with previously installed versions. The operations are performed using AnswerModules Administration tools available within the Content Server Admin pages.

Upgrading on a Clustered Environment

When upgrading a Module Suite installation on a clustered Content Server environment, the overall procedure will vary.

In a clustered environment it is **mandatory** to install the Module Suite components on all nodes, but it is important to notice that the single installation steps must not be performed on each single node separately, as certain operations already affect the whole cluster.

At a high level, the suggested procedure is to perform a complete the upgrade procedure on the primary node of the cluster, and then reconcile the remaining nodes.

Please refer to the Upgrading of a clustered environment guide for detailed info.

Upgrading Script Console

Script Console can be upgraded performing a so-called "parallel" upgrade, which means installing on the same/different server the newer version of the console and configure it as the previous one.
This typically requires to copy over the relevant configuration files from the previous Script Console together with any custom script you might have created/deployed on the console: %SCHOME%/config/cs-console-schedulerConfiguration.xml, %SCHOME%/config/cs-console-security.xml %SCHOME%/config/cs-console-systemConfiguration.xml

Prerequisites¶

This guide assumes certain resources to be readily available while performing the installation. Please ensure the following have been provisioned before starting the installation process:

- ☑ Admin-level access to the servers on which the software will be installed
- ☑ Admin user access to the Content Server instance.
- ☑ The Module Suite **installers** or installation packages compatible with the target environment

Installer versions

Before proceeding with the installation, make sure that the installer version matches the OpenText Content Server target system version.

E.g.:

- ***module-suite-2.7.0-OTCS162.exe*** is the Windows installer for OpenText Content Server 16.2.X;

- *module-suite-2.6.0-OTCS162.exe* is the Windows installer for OpenText Content Server 16.2.X;
- *module-suite-2.5.0-OTCS162.exe* is the Windows installer for OpenText Content Server 16.2.X;
- *module-suite-2.4.0-OTCS16.exe* is the Windows installer for OpenText Content Server 16.0.X;
- *module-amcontentscript-2.3.0-OTCS105.exe* is the Windows installer for OpenText Content Server 10.5.X;
- *module-amcontentscript-2.2.0-OTCS10.exe* is the Windows installer for OpenText Content Server 10.0.X;

Note: Starting with version 3.2.0, the OTCS identifier (OTCS10, OTCS105, OTCS162 ...) is no longer present in the installer names.

☑ A valid AnswerModules **activation key**, either in plain text format or in OTCS Configuration Export XML format. The latter is the suggested option as it will prevent errors due to manual input.

Activation keys must match Module Suite version

Module Suite activation keys are specific to a target software version. i.e. An Activation key intended for Module Suite version 3.1 **will not be valid** on Module Suite version 3.2

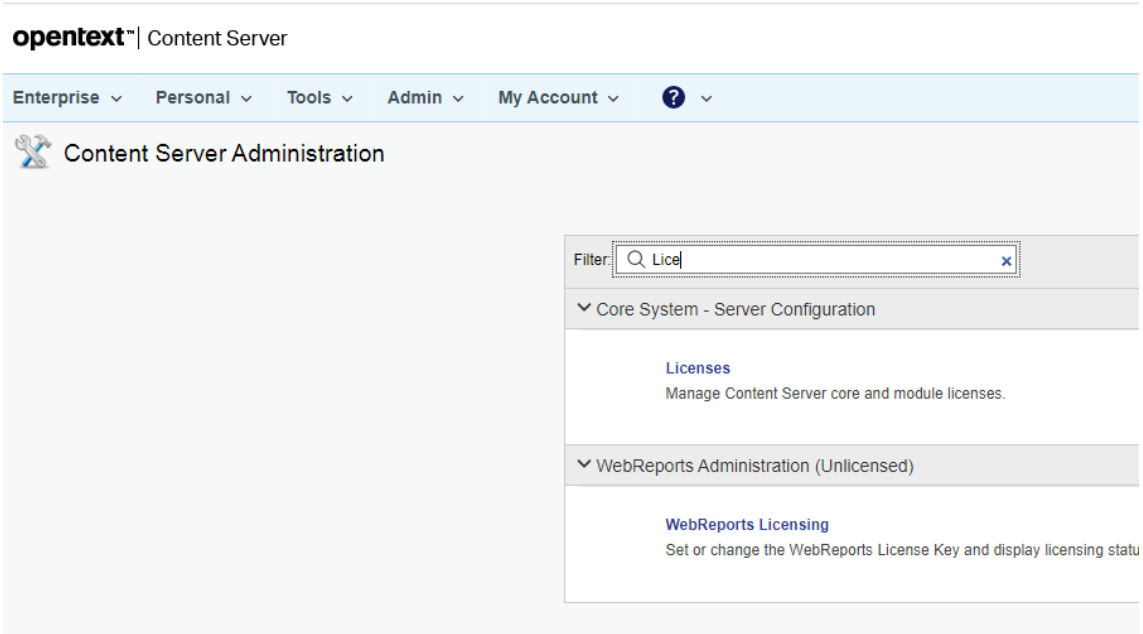
Keys and System Fingerprint

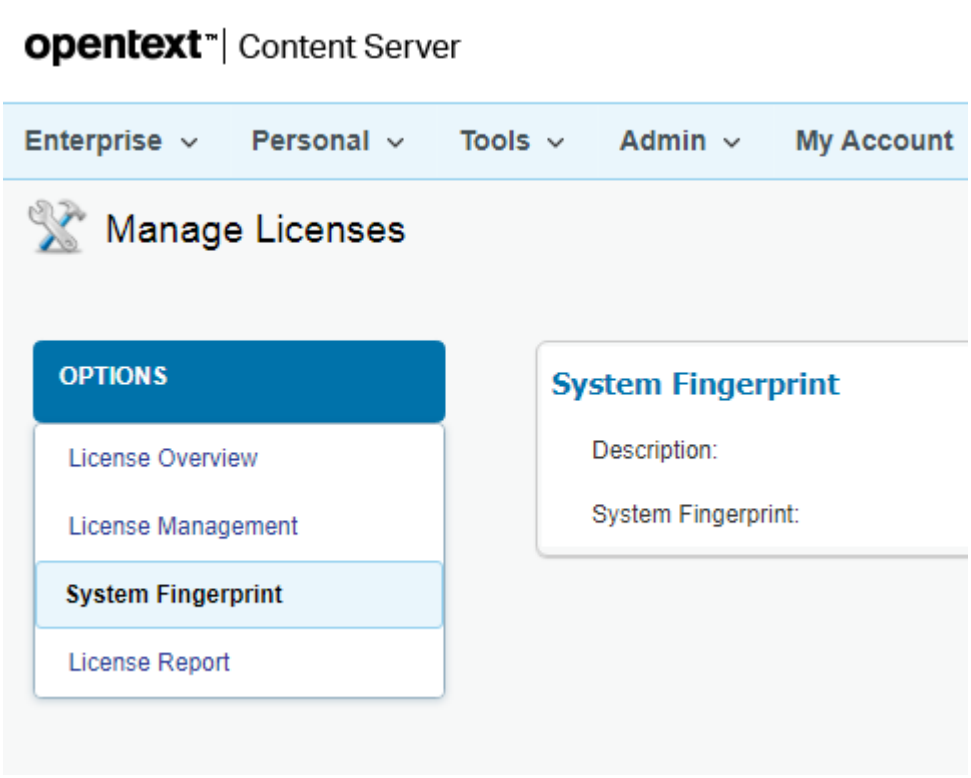
- An activation key is only required starting from version 1.7.0 of the Module Suite.
- Starting from version 2.0.0 activation keys are bound to the system's fingerprint.

How do I get an activation key?

In order to activate Module Suite you need a valid activation key. Activation keys can be requested to AnswerModules Support *(https://support.answermodules.com)* by providing the OpenText Content Server System Fingerprint.

You can read your's environment fingerprint from the OpenText Admin Pages as shown below





- [x] Any relevant **hotfixes** released for the Module Suite version being installed

Hotfixes

Hotfixes and patches are continuously published on the AnswerModules Support Portal. Check the availability of applicable patches when starting a new installation.

E.g. https://support.answermodules.com/portal/kb/articles/2-2-0-patches-and-hotfixes-for-content-server-16 *(https://support.answermodules.com/portal/kb/articles/2-2-0-patches-and-hotfixes-for-content-server-16)*

Next Steps

Once all the prerequisites are met, please proceed to the **Upgrade** guide:

upgrade

Upgrading Module Suite¶

This guide covers the step-by-step procedure to perform an upgrade of a Module Suite installation.

Check prerequisites

The guide assumes that the prerequisites to perform the upgrade operation are met. If that is not the case, please go back to the Upgrade overview.

Deploy the new Modules on the target system¶

During this phase, the updated Module binaries will be deployed on the target system. The steps to perform for the deployment are exactly the same as the ones covered during a clean installation. Depending on the target platform, refer to one of the following resources:

- if you are installing on a Windows environment: Deploy on Windows
- if you are installing on a Unix/Linux environment: Deploy on Unix/Linux

Perform the Module upgrade¶

This phase is roughly equivalent to the Module installation phase performed upon initial Module Suite installation. The difference is that the system will already include an older version of the Modules, which will have to be replaced.

In order to proceed with the upgrade of the modules, please follow these steps:

- ☑ Start the Content Server services
- ☑ Login as Administrator and access the Module administration panel
- ☑ Access the Content Server Admin pages > **Core System - Module Configuration** > **Upgrade Modules**
- ☑ From the available modules, select **“AnswerModules Content Script x.y.z”**
- ☑ Follow the installation steps and restart Content Server when prompted.
- ☑ Access the Content Server Admin pages > **Core System - Module Configuration** > **Upgrade Modules**
- ☑ From the available modules, select **“Answer Modules - Beautiful Web Forms x.y.z”**
- ☑ Follow the installation steps and restart Content Server when prompted.
- ☑ Access the Content Server Admin pages > **Core System - Module Configuration** > **Upgrade Modules**
- ☑ From the available modules, select **“Answer Modules - Smart Pages x.y.z”**
- ☑ Follow the installation steps and restart Content Server when prompted.
- ☑ At this point, the upgraded Modules have been installed in the Content Server system and have replaced the older versions.

Apply the available hotfixes¶

- [x] Stop Content Server
- [x] Apply relevant hot fixes
- [x] Start Content Server

Activate the software¶

- [x] Activate the software by applying the new software activation key. Depending on the format in which the key was provided, you can use one of the following approaches:
  - If you plan to apply the license key manually, please proceed to Activation through manual key setup.
  - Alternatively, if you plan to import the licensing configuration settings, please proceed to the Activation through key import.

Upgrading from Versions Below 3.2¶

> **Critical Upgrade Step**
>
> If you are upgrading from a version below 3.2, it is crucial to perform the following step before importing the new libraries:
>
> Rename `CSSystem` to `_CSSystem_` in the Content Script Volume.

Step-by-Step Renaming Process¶

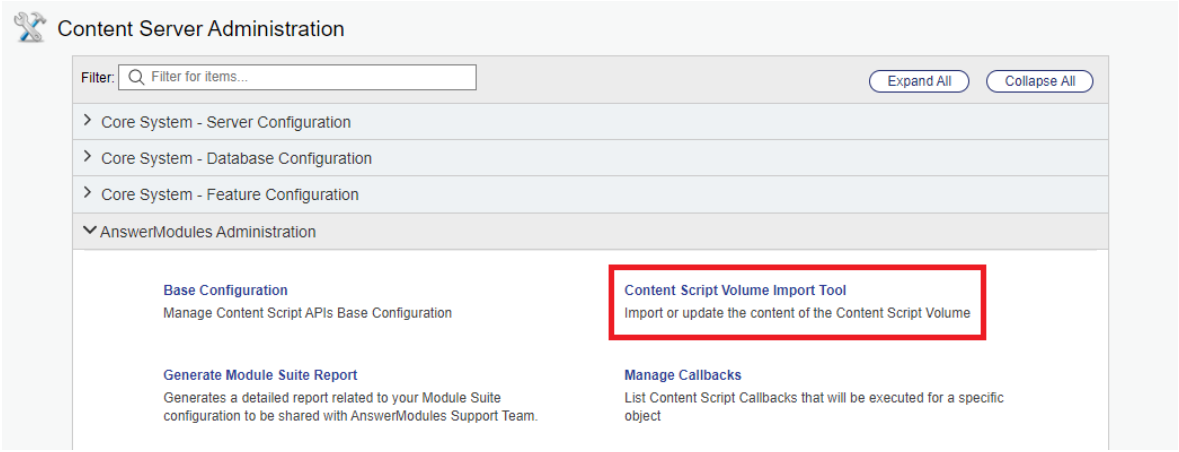
1. **Locate the Content Script Volume**: Navigate to your Content Script Volume in the system.
2. **Find the CSSystem**: Identify the `cssystem` within the Content Script Volume.
3. **Perform the Rename**: Change the name from `cssystem` to `csystem`.
4. **Verify the Change**: Double-check that the rename operation was successful.

Update the Module Suite Configuration¶

Using the Content Script Volume Import Tool, check for the presence of updates or conflicts in the System library.

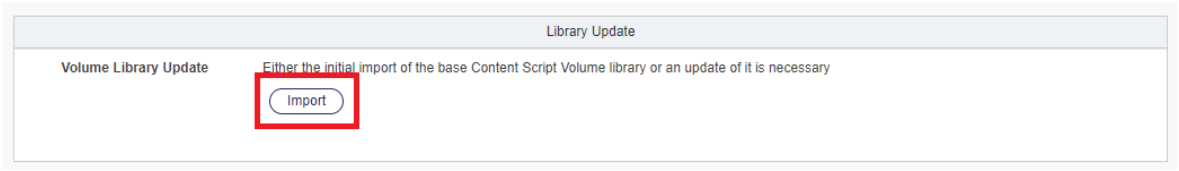
- [x] As the system Admin user, open the Content Server Administration pages.

- [x] Locate the **AnswerModules Administration** section. Within this section, open the **Content Script Volume Import** tool.



- [x] Within the Content Script Volume Import page, locate the **Library Update** section (it can be found at the very top of the page).

  Click the **Import** button.

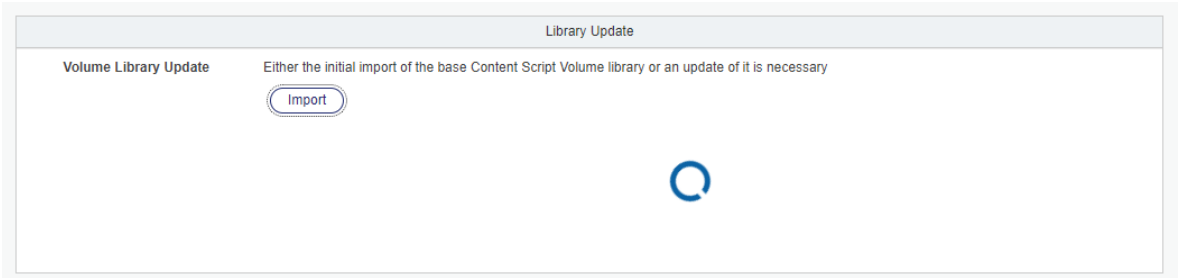


What if the Library Update section is not present?

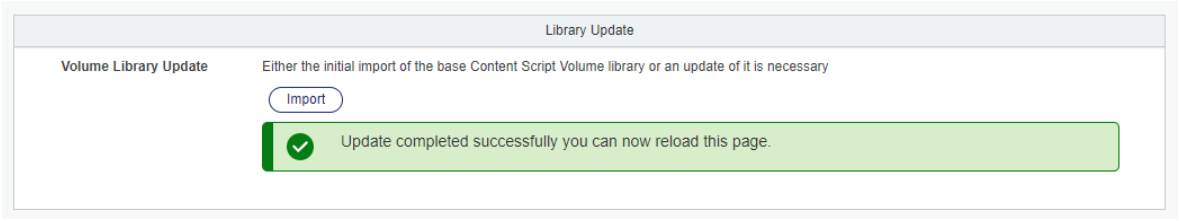
The **Library Update** section at the top of the import tool will only show up if there are updates to be applied to the System libraries.

If none are present, this section will not be found and the step can be skipped.

- [x] Wait for the import operation to complete.



- [x] Once the operation is complete, a success message will be shown.



Upon refreshing the page, the **Library Update** section will no longer be visible. This indicates that all relevant libraries have been imported.

- [x] Within the Content Script Volume Import page, locate the **Volume's Conflicts Resolution** section. The section may take some time to process the status of the Volume. In this case, it will show as loading.

- [x] Check for any outstanding conflicts and optional updates and update as needed.

Refer to the Content Script Volume Import Tool for further details on conflicts and conflict resolution.

Custom changes to library components

In case any of the standard components were customized, patched or otherwise modified, or new custom components were added within the standard library, make sure that you transfer any relevant changes to the new libraries before deleting the old version.

What do I need to upgrade ?

How the library upgrade works¶

The 'Upgrade' operation will rename the existing library folders in the Content Script volume, and import a new version of the same (the only exception is the 'CSFormTemplates' folder, which will be discussed later). As such, any modification that has been applied to one of the libraries will be relocated and no longer available.

Examples include:

- any custom Beautiful WebForms components added to the CSFormSnippets folder
- any custom Rest API endpoints added to the CSServices folders
- any callbacks configured in the CSEvents or CSSynchEvents folders
- any Classic UI modifications applied through the CSMenu, CSAddItems, CSBrowseView, CSBrowseViewColumns
- any other object created or modified within one of the upgraded folders

As part of the upgrade operation, you should identify such changes and make sure they are ported to the new libraries.

CSFormTemplates have a slightly different upgrade process. Since objects in this folder are referenced by object DataID (their unique identifier on OTCS)they can't be replaced with the updated version, since this would potentially cause issues in any existing form using the template. For this reason, the upgrade process for CSFormTemplates automatically updates each single template by adding a new version to the object, thus preserving the original DataID. For this reason, no "backup" folder will be found for CSFormTemplates..

- [x] Cleanup. The folders named “Backup-_yyyyMMdd-AAAAAA” are backup folders containing the previously installed library scripts/snippets. They can be safely exported and removed

clustered installation installation upgrade

Upgrading Module Suite on a clustered environment¶

In a Content Server clustered environment, it is mandatory to install Module Suite modules on each node that makes up the cluster.
The installation process in a cluster is more complex than installing on a single server, as a slightly different procedure must be performed on each remaining node in the cluster after installing the modules on the first one. The recommended approach is to install Module Suite on a **primary node** (the node on which the primary OpenText Admin Content Server services are installed and configured) and then copy the installed modules to each other node (**secondary nodes**) in the cluster. This approach ensures that all installed modules are identical and that the patch level on all nodes is the same.

We will refer to the Content Server installation directory as **%OTCS_HOME%**.

Deployment on the primary node¶

Module Suite package upgrade on a Primary node is identical to the upgrade process into the non-clustered environment.

- ☑ Stop Content Server services on all the nodes in the cluster
- ☑ Proceed with the Module Suite upgrade on the **Primary node**

  Detailed description of this procedure can be found in Upgrading Module Suite guide.

Deployment on the secondary node(s)¶

Once the Module Suite modules are upgraded on the primary node, the module packages can be deployed on the remaining cluster nodes.
Proceed with the following upgrade steps on all **Secondary nodes**

- ☑ Make a copy of the following resources and make them available in a working folder on the Secondary node:

  1. %OTCS_HOME%/module/anscontentscript_x_x_x
  2. %OTCS_HOME%/module/ansbwebform_x_x_x
  3. %OTCS_HOME%/module/anscontentsmartui_x_x_x
  4. %OTCS_HOME%/support/anscontentscript
  5. %OTCS_HOME%/support/ansbwebform
  6. %OTCS_HOME%/support/anscontentsmartui

  Installed Modules

  The actual folders to be copied depend on the Module that have been installed on the Primary node.
  e.g. if you are only installing Content Script and Beautiful WebForms modules, the "anscontensmartui" folders will not be available in the Primary node.

- ☑ On the Secondary node, ensure that all Content Server services are stopped.
- ☑ On the Secondary node, move the following folders and all their content to a backup folder

  1. %OTCS_HOME%/module/anscontentscript_x_x_x
  2. %OTCS_HOME%/module/ansbwebform_x_x_x
  3. %OTCS_HOME%/module/anscontentsmartui_x_x_x
  4. %OTCS_HOME%/support/anscontentscript
  5. %OTCS_HOME%/support/ansbwebform
  6. %OTCS_HOME%/support/anscontentsmartui

- [ ] On the Secondary node, copy all the upgraded modules and support folders previously identified to their corresponding target location within %OTCS_HOME%.

- [ ] On the Secondary node, proceed to manually **reconcile the opentext.ini** file in **%OTCS_HOME%/config**.
  Pay particular attention to the [Modules] and [javaserver] sections on the opentext.ini file.

- [ ] On the Secondary node, start the Content Server services.

Other installation guides

installation

Installing Content Script¶

This guide is specific to the installation of the Content Script component of Module Suite.

Module Suite installation

If you are interested in installing the full Module Suite, including all its components, please follow the Installing Module Suite guide.

In order to perform the installation of the Content Script module, you will have to follow a similar procedure to the one described in Installing Module Suite, with the following exceptions:

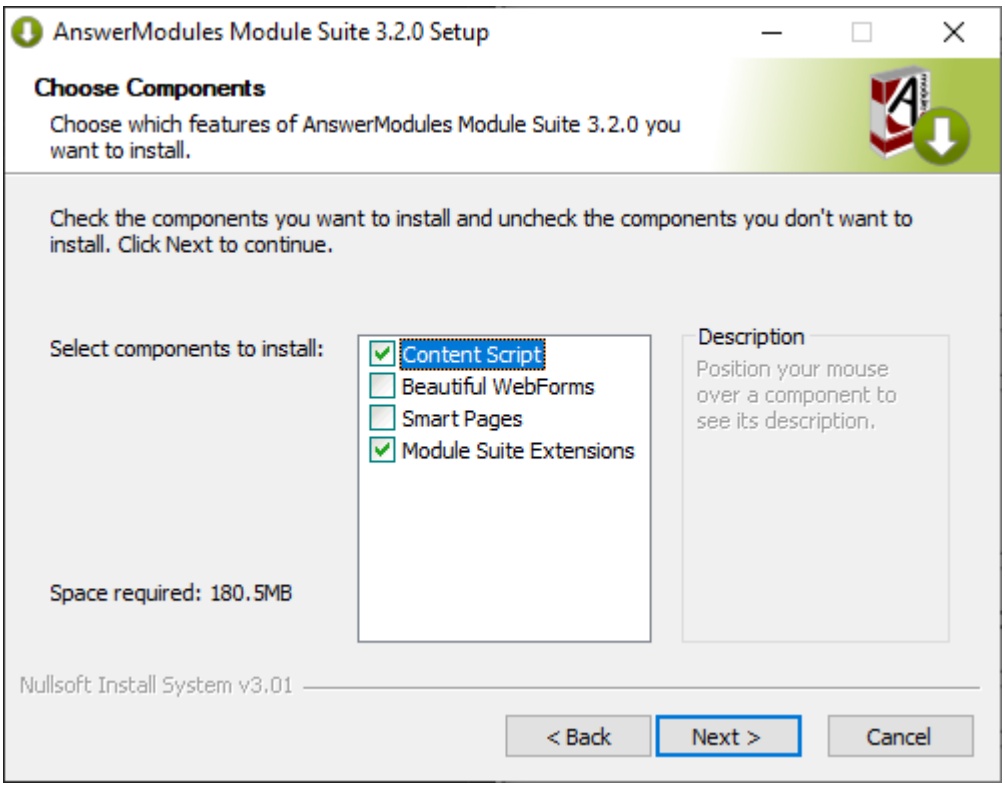
Deployment Phase - Select the components to be installed¶

Reference in Module Suite installation guide

This entry refers to the following instructions: Installing Module Suite - Deploy - Step by step deployment

When prompted to select the Module Suite components to install, only select the required options:

- Content Script
- Module Suite Extension Packages (optional)



Installation Phase - Step-by-step Installation¶

Reference in Module Suite installation guide

This entry refers to the following instructions: Installing Module Suite - Install - Step by step installation

When accessing the "Install Modules" administration panel, the only available Module to install will be "AnswerModules Content Script x.y.z".

Follow the installation steps for this module and restart when prompted.

installation

Installing Beautiful WebForms¶

This guide is specific to the installation of the Beautiful WebForms component of Module Suite.

Module Suite installation

If you are interested in installing the full Module Suite, including all its components, please follow the Installing Module Suite guide.

Prerequisite : Content Script

The Beautiful WebForms Modules has a dependency on the Content Script engine, which is part of the Content Script Module. Content Script engine has to be installed and properly configured (including activation) in order to proceed with the standalone installation of the Beautiful WebForms Module.

In order to perform the installation of the Beautiful WebForms module, you will have to follow a similar procedure to the one described in Installing Module Suite, with the following exceptions:

Getting Started - Prerequisites¶

The Beautiful WebForms Modules has a dependency on the Content Script engine, which is part of the Content Script Module.

Since the Content Script Module will already be installed and configured on your system, you will not require a separate **Activation key** to proceed with the standalone installation of the Beautiful WebForms Module.

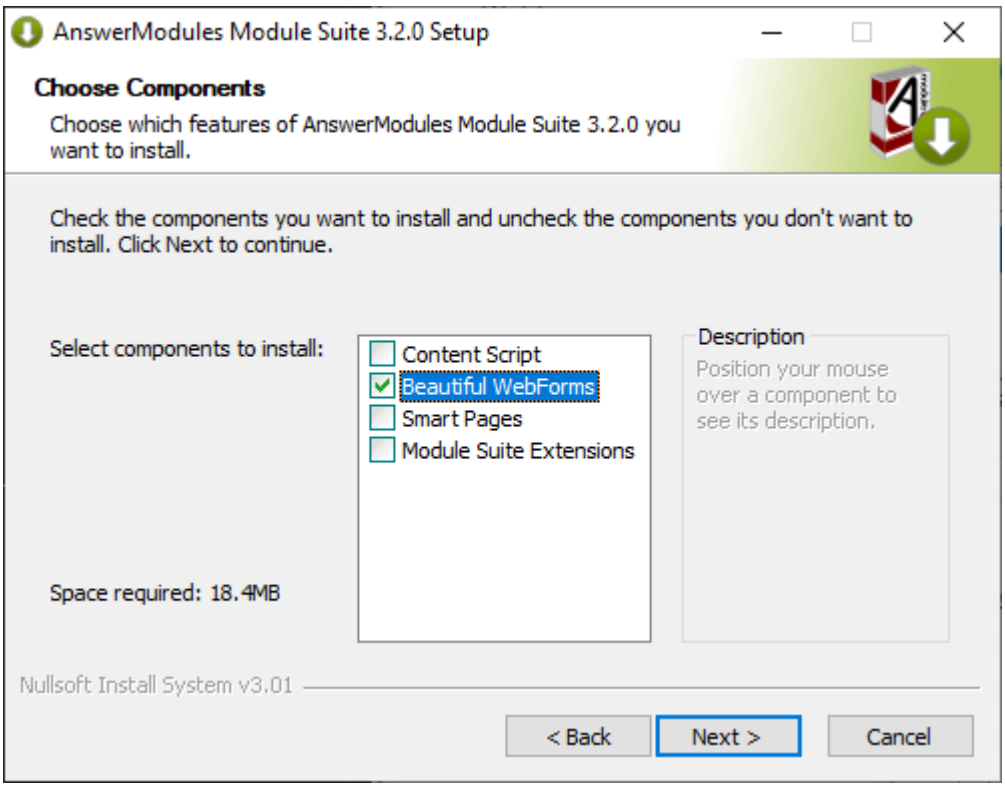
Deployment Phase - Select the components to be installed¶

Reference in Module Suite installation guide

This entry refers to the following instructions: Installing Module Suite - Deploy - Step by step deployment

When prompted to select the Module Suite components to install, only select the required options:

- Beautiful WebForms



Installation Phase - Step-by-step Installation¶

Reference in Module Suite installation guide

This entry refers to the following instructions: Installing Module Suite - Install - Step by step installation

When accessing the "**Install Modules**" administration panel, the only available Module to install will be "**AnswerModules Beautiful WebForms x.y.z**".

Follow the installation steps for this module and restart when prompted.

Activation Phase¶

The activation phase can be skipped when performing a standalone installation of the Beautiful WebForms Module. Module Suite Activation is associated with the Content Script installation, which is a prerequisite.

installation

Installing Smart Pages¶

This guide is specific to the installation of the Smart Pages component of Module Suite.

Module Suite installation

If you are interested in installing the full Module Suite, including all its components, please follow the Installing Module Suite guide.

Prerequisite : Content Script

The Smart Pages Modules has a dependency on the Content Script engine, which is part of the Content Script Module. Content Script engine has to be installed and properly configured (including activation) in order to proceed with the standalone installation of the Smart Pages Module.

In order to perform the installation of the Smart Pages module, you will have to follow a similar procedure to the one described in Installing Module Suite, with the following exceptions:

Getting Started - Prerequisites¶

The Smart Pages Modules has a dependency on the Content Script engine, which is part of the Content Script Module.

Since the Content Script Module will already be installed and configured on your system, you will not require a separate **Activation key** to proceed with the standalone installation of the Smart Pages Module.

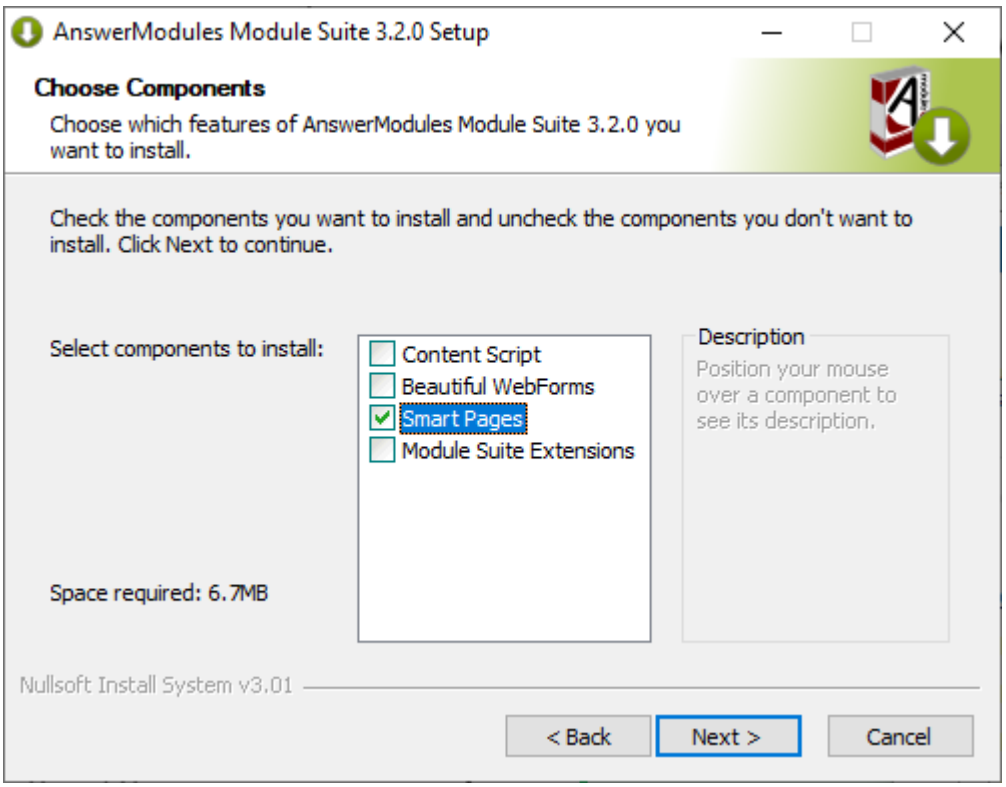
Deployment Phase - Select the components to be installed¶

Reference in Module Suite installation guide

This entry refers to the following instructions: Installing Module Suite - Deploy - Step by step deployment

When prompted to select the Module Suite components to install, only select the required options:

- Smart Pages



Installation Phase - Step-by-step Installation¶

Reference in Module Suite installation guide

This entry refers to the following instructions: Installing Module Suite - Install - Step by step installation

When accessing the "Install Modules" administration panel, the only available Module to install will be "AnswerModules Smart Pages x.y.z".

Follow the installation steps for this module and restart when prompted.

Activation Phase¶

The activation phase can be skipped when performing a standalone installation of the Smart Pages Module. Module Suite Activation is associated with the Content Script installation, which is a prerequisite.

installation unix

Script Console installation guide¶

Installation procedure¶

JVM and Servlet API

Ensure that the Script Console runs on the same JVM version as the OpenText Content Server where the corresponding version of the Content Script Module is installed.

For instance:

- If you're installing **Script Console 3.0** and you have **Content Script 3.0** on **OpenText Content Server 20.2**, use the **identical JVM version** that the OpenText Content Server environment utilizes.

When setting up the Script Console to run as a web application, note the following:

- **Script Console** is designed for **Servlet-API 3.X**.
- It is also compatible and can run on **Servlet-API 4.x** without issues.

Script Console can be configured to run in different modes. Common scenarios are:

1. standalone interactive console, connected to OTCS: mainly used for batch processing and administration tasks
2. standalone script interpreter, connected to OTCS: mainly used for scheduling administration tasks
3. standalone lightweight webserver (based on embedded application server), connected or not connected to OTCS
4. web application deployed on external application server, connected or not connected to OTCS

This guide covers the standard installation procedure of the Content Script Console (standalone based on embedded application server) which is compliant with the options 1, 2 and 3 of the above list.

For alternative deployment scenarios, including deployment on an external application server, please make reference to AnswerModules Support Team and guides available through Support Portal.

- [x] Run the **Script Console Installer** (WINDOWS), or extract the Script Console archive, and install the Script Console in your favourite location (this step should be executed by an user having local administrative privileges)

Environment variables

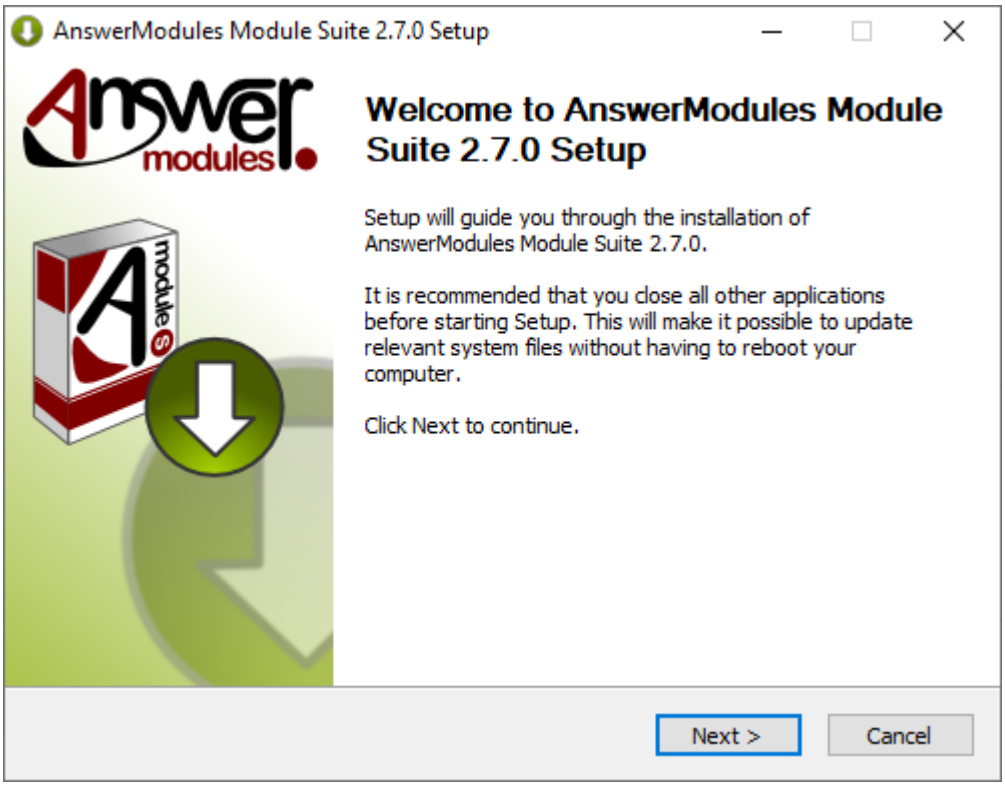
The Script Console requires an environment variable to be defined in order to work properly, for your convenience this variable is automatically defined on windows server by the Script Console installer:

- **AM_CONSOLE_DATA**: the Script Console's root folder

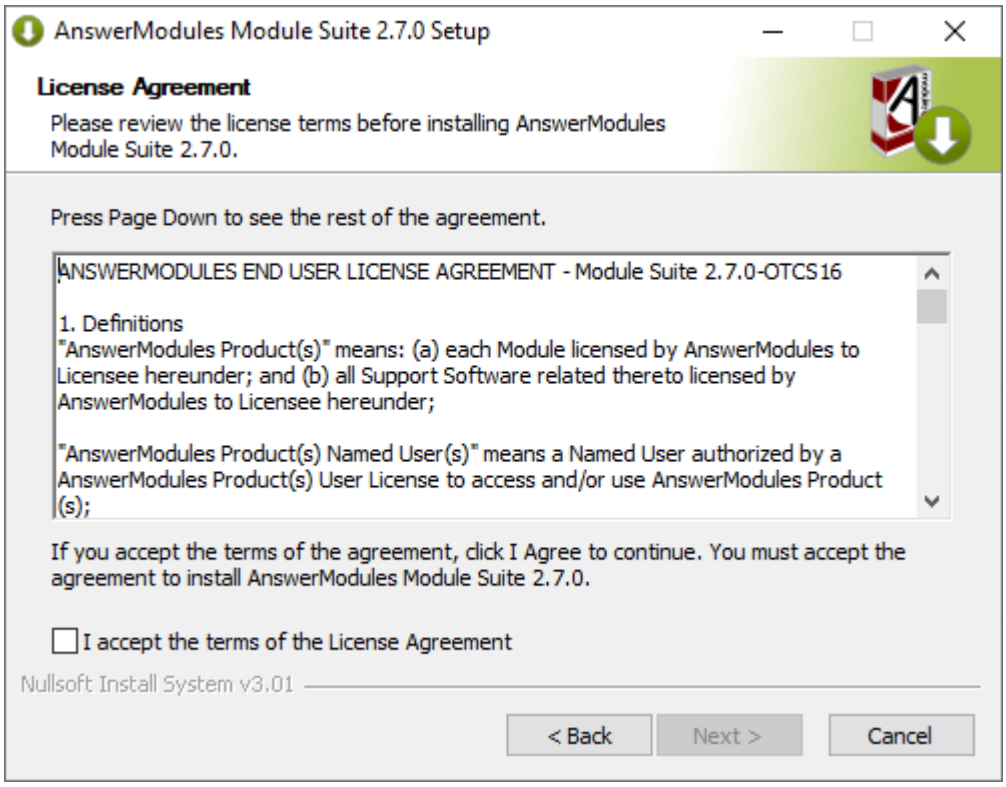
Step-by-Step procedure

The following screens will guide you through the deployment of Script Console runtime.

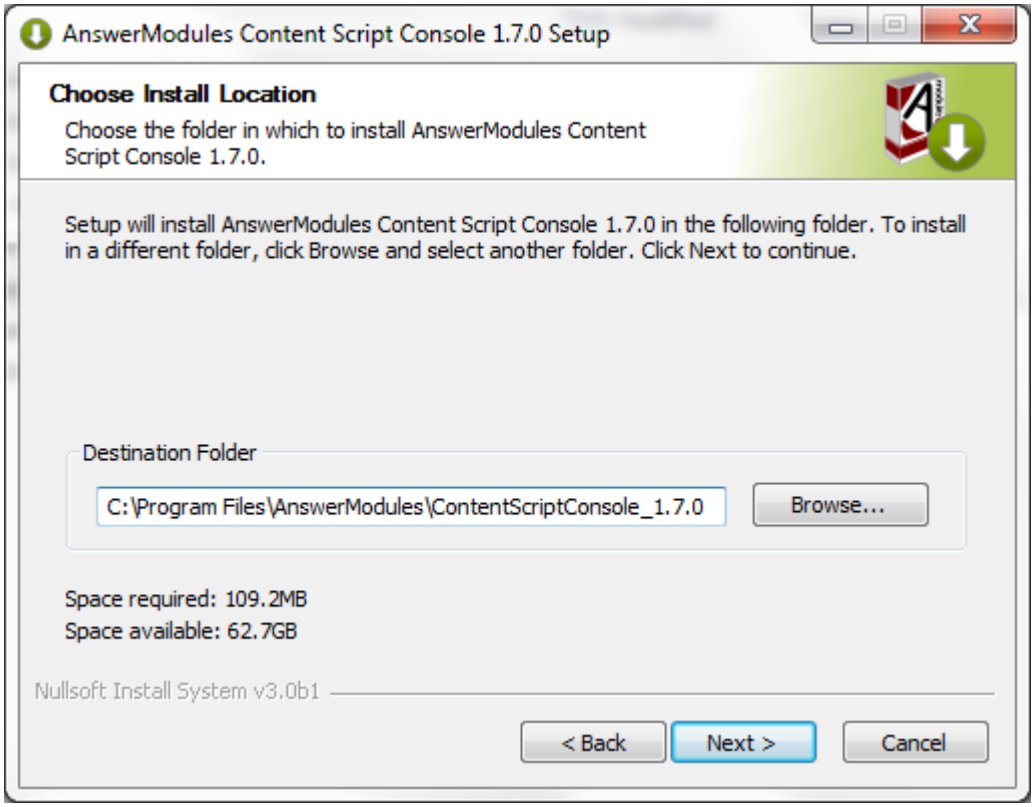
1. Welcome Screen: Select “Next” when ready to start the installation.



2. EULA Screen: Acceptance of the end-user license agreement is mandatory for proceeding with the installation
   A copy of the agreement will be available, upon installation, in:
   %AM_CONSOLE_DATA%/license/EULA Select “Next” when ready.



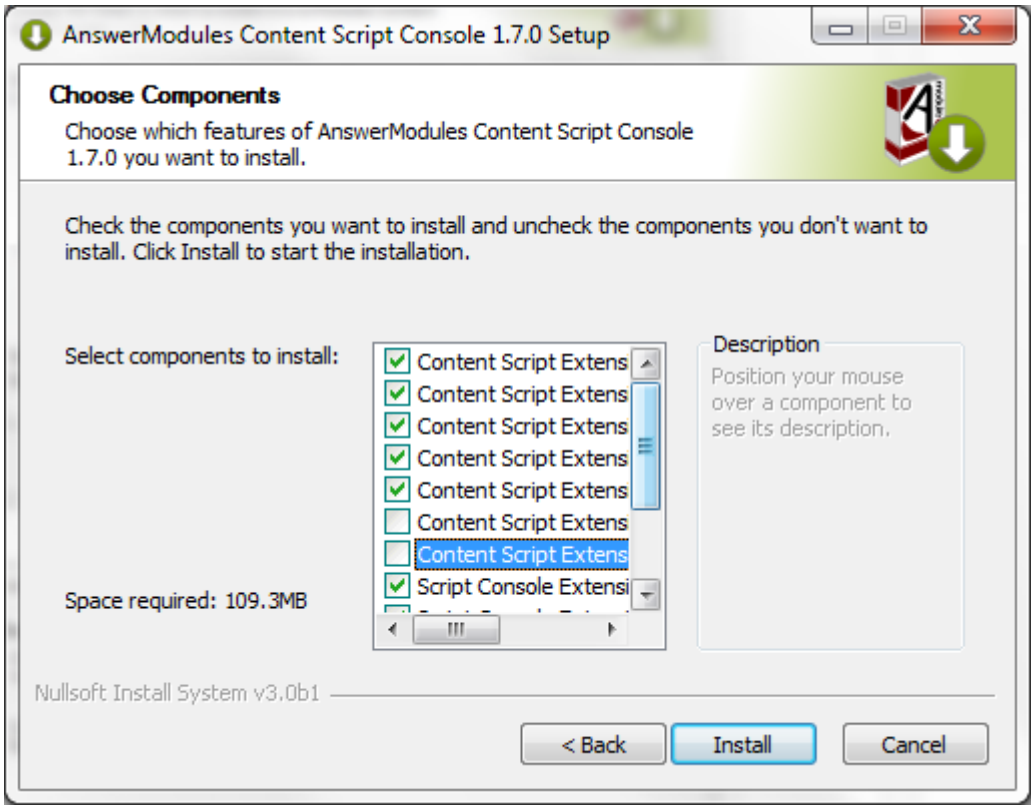
3. AM_CONSOLE_DATA selection: Choose the location where the Script Console components will be installed.
   E.g.
   `E:\AM\SC_2_7_0\`



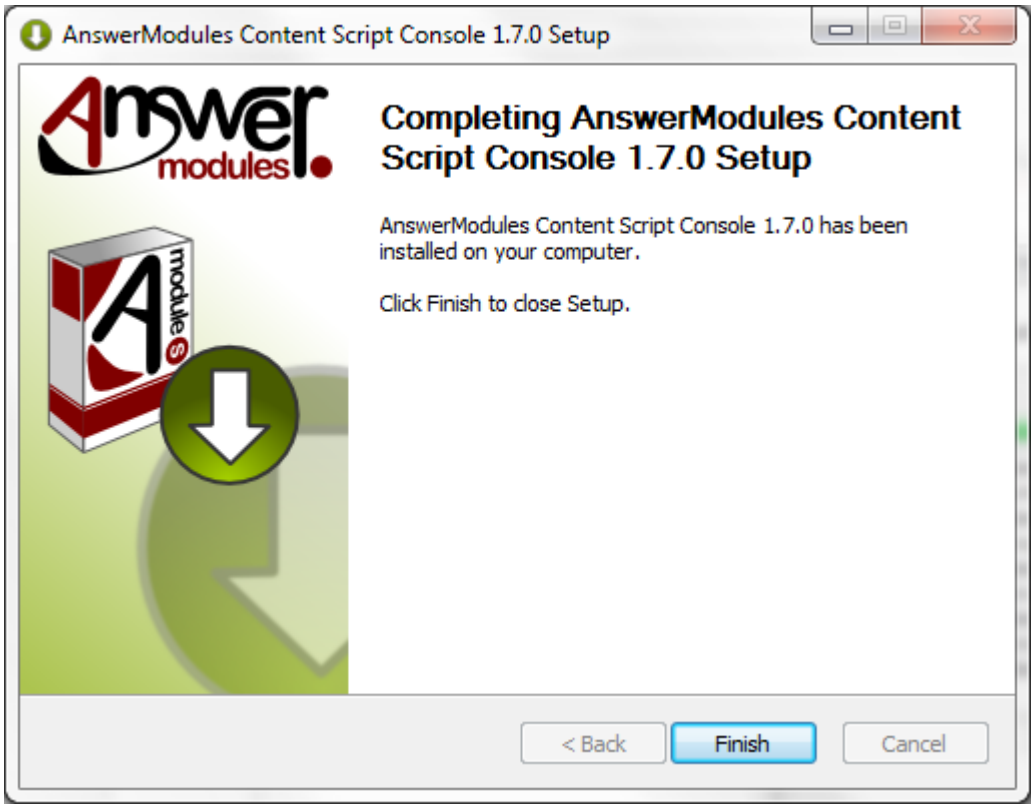
4. Script Console Application and Content Script Extension Packages: there are two different types of extensions that can be installed:

5. Content Script Extensions are extensions for the embedded Content Script Engine.

6. Script Console Applications



11. Installation completed: Select “Finish” and return to the installation checklist to finalize the module setup.



- [x] Copy Content Server's libraries to the Script Console runtime

Content Server libraries required

Some Content Script extension packages require additional Java libraries that are specific to the target Content Server environment, and are not distributed with the module. The required library files are:

- csapi.jar
- service-api-X.X.XX.jar

and can be found in the web app located in:

`%OTCS\_HOME%\\webservices\\java\\webapps\\cws.war`

- classificationsservice-api-X.X.XX.jar

which can be found in the web app located in:

`%OTCS\_HOME%\\webservices\\java\\webapps\\cs-services-classifications.war`

- physicalobjectsservice-api-X.X.XX.jar

which can be found in the web app located in:

`%OTCS\_HOME%\\webservices\\java\\webapps\\cs-services-physicalobjects.war`

- recordsmanagementservice-api-X.X.XX.jar

which can be found in the web app located in:

`%OTCS\_HOME%\\webservices\\java\\webapps\\cs-services-recordsmanagement.war`

- oml.jar

which can be found in: `%OTCS\_HOME%\\ojlib`

To retrieve the files:

- copy the file named XXX.war to a temporary folder
- rename the file XXX.war in XXX.zip
- extract the zip archive contents locate the files in the WEB-INF/lib folder

Once the files have been located, copy them to the folder: %AM_CONSOLE_DATA%/runtime/amlib

Copy libraries form Content Script

All the libraries mentioned above but ** oml.jar ** are usually also found in the installation folder of the Content Script module: `%OTCS_HOME/module/anscontentscript_X_Y_Z/amlib`

- [x] Perform basic configuration of the Script Console. The main configuration file is located in: %AM_CONSOLE_DATA%/config/cs-console-systemConfiguration.xml
Default configuration will be similar to the following:

```
<config>
    <systems>
        <system id="TEST">
            <name>Content Server TEST Environment</name>
            <serverHost>localhost</serverHost>
            <serverPort>2099</serverPort> <!-- OTCS API port if operating as connected -->
            <!-- <OTHome>/opt/am/sc/</OTHome>  Script Console Home folder. -->
```

```
                <local-repository-home>TEST</local-repository-home>
                <local-repository-encoding>UTF-8</local-repository-encoding>
                <otcs-repository-encoding>UTF-8</otcs-repository-encoding>
                <systemVars>
                    <systemVar name="img">/csconsoleimg/</systemVar><!-- Base path for static c
                    <systemVar name="url">/csconsole</systemVar><!-- Base path for scripts cont
                    <systemVar name="csModulePath"></systemVar><!-- Base path for runtime conte
                    <systemVar name="clientIPAddress"></systemVar><!-- Actual IP Address to be
                </systemVars>
                <serviceVars>
                    <serviceVar service="core" name="amcs.core.httpProxyHostname" ></serviceVar
                    <serviceVar service="core" name="amcs.core.httpProxyPort">80</serviceVar>
                    <serviceVar service="core" name="amcs.core.httpProxyUsername"></serviceVar>
                    <serviceVar service="core" name="amcs.core.httpProxyPassword"></serviceVar>
                    <serviceVar service="core" name="amcs.core.httpMaxConnPerRoute">20</service
                    <serviceVar service="core" name="amcs.core.httpMaxConnTotal">50</serviceVar
                    <serviceVar service="core" name="amcs.core.httpOTCSSchema">http</serviceVar
                    <serviceVar service="core" name="amcs.core.tempFilePath">/tmp/</serviceVar>
                    <serviceVar service="dbx" name="amcs.dbx.activeProfiles">default</serviceVa
                    <serviceVar service="dbx" name="amcs.dbx.cacheClients">false</serviceVar>
                    <serviceVar service="dbx" name="amcs.dbx.appKey.default"></serviceVar>
                    <serviceVar service="dbx" name="amcs.dbx.appSecret.default"></serviceVar>
                    <serviceVar service="dbx" name="amcs.dbx.authToken.default"></serviceVar>
                    <serviceVar name="ans.appbuilder.requestContext" service="appbuilder">/otcs
                    <serviceVar name="ans.appbuilder.supportContext" service="appbuilder">/img/
                    <serviceVar name="ans.appbuilder.httpOTCSSchema" service="appbuilder">http<
                </serviceVars>
                <users>
                    <user password="B594193ED65B934A5D11E5DE2323131E8C70" username="Admin"/>
                </users>
            </system>
        </systems>
        <extensions id="forms">
            <repositories>
                <repository commands="false" encoding="UTF-8" home="forms" root="scripts/ext" s
            </repositories>
        </extensions>
    </config>
```

The base configuration allows to specify one or more “system” objects which represent OTCS instances to which the console will be able to connect.

How to setup your base configuration

The base configuration can be edited manually, or, alternatively, configuration parameters can be downloaded from a target Content Server instance. This feature comes particularly handy for installations that include multiple Content Script Extension Packages, each with its own configuration settings.

- [x] Apply any available hotfix(es)

Hot to install a hotfix

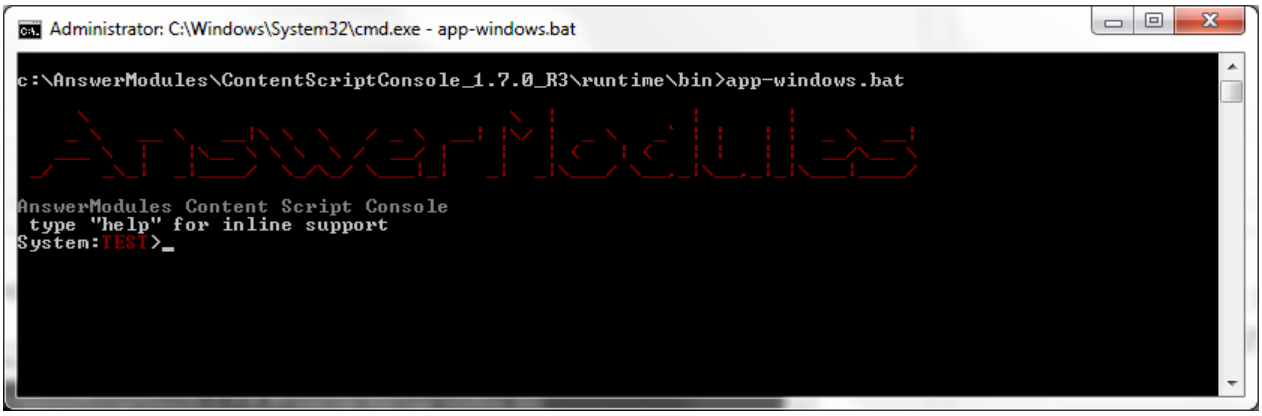
Before you install any hotfix, please backup all essential files. To install the hotfix, download the hotfix from the Support portal and save it to a temporary location. Make sure Script Console services (or executable) are completely stopped. From the temporary location, extract the contents of the hotfix to the <Script_Console_home> directory and then restart it.
The directory (directories) and file(s) contained in the hotfix(es) you install will be copied to <Script_Console_home>

Please always make reference to the hotfix's description file:/hotfixes/hotFix_ANS_XXX_YYY_ZZZ.hfx for specific installation instructions or pre/post installation procedures

Configure Script Console¶

To perform configuration against an OTCS instance, run the Script Console in shell mode. To do so, open a Windows Commands Processor and move to the folder: **%AM_CONSOLE_DATA%/runtime/bin** which includes the Script Console's executables scripts

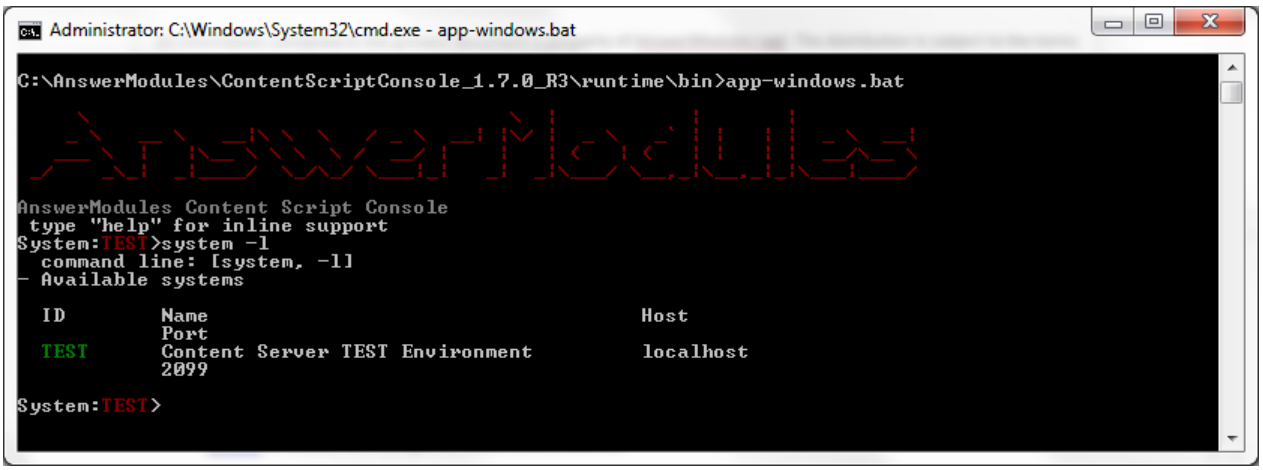
- Run the `app-windows.bat` or `app.sh` script
- The following prompt should appear:



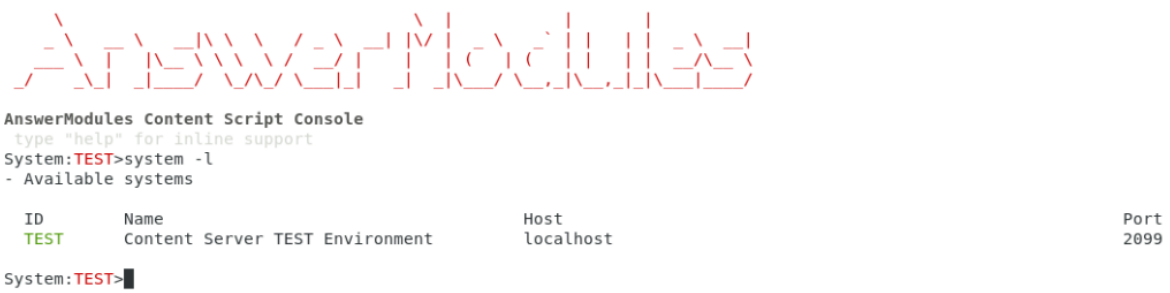
Unix

```
centos:/opt/am/sc/runtime/bin$ export AM_CONSOLE_DATA=/opt/am/sc
centos:/opt/am/sc/runtime/bin$ ./app.sh
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.

AnswerModules Content Script Console
 type "help" for inline support
System:TEST>
```

- The default TEST system is selected. To list all available systems, use the system command with the list flag (`-l, --list`). E.g. `system -l`:



Unix



- To create a new system (for example, *LOCAL*) use the system command with the add flag (`-a, --add`) followed by the ID of the new system. E.g. `system -a LOCAL`

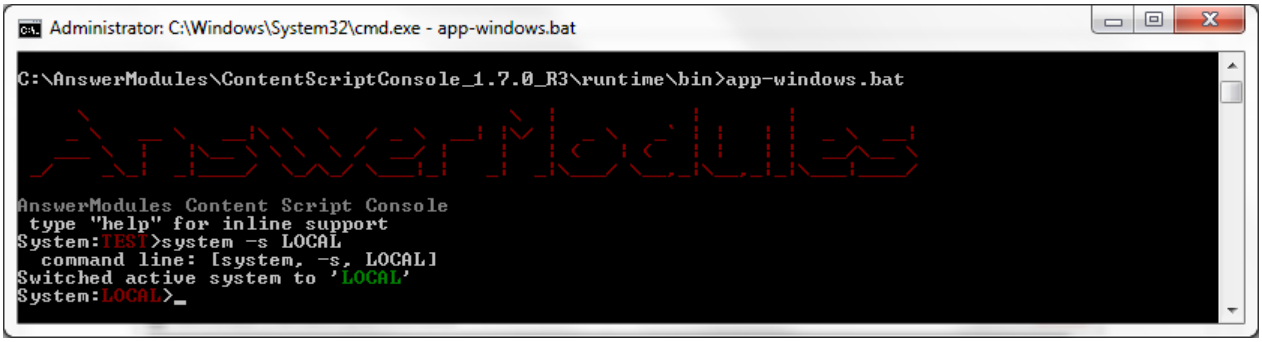
  The shell will prompt for the required base values, such as `hostname` and `port` number.

Unix

```
System:TEST>system -a LOCAL
  Host: localhost
  Port: 2099
  Username (opt): Admin
  Password (opt): ************
- New System details
  ID:   LOCAL
  NAME: Default
  HOST: localhost
  PORT: 2099

Shutting down..
centos:/opt/am/sc/runtime/bin$
```

Upon creating a new system, the Script Console will require a restart and will automatically shutdown.

- Switch the active system to LOCAL using the system command with the system flag (`-s`) followed by the ID of the target system. E.g. `system -s LOCAL`



Unix

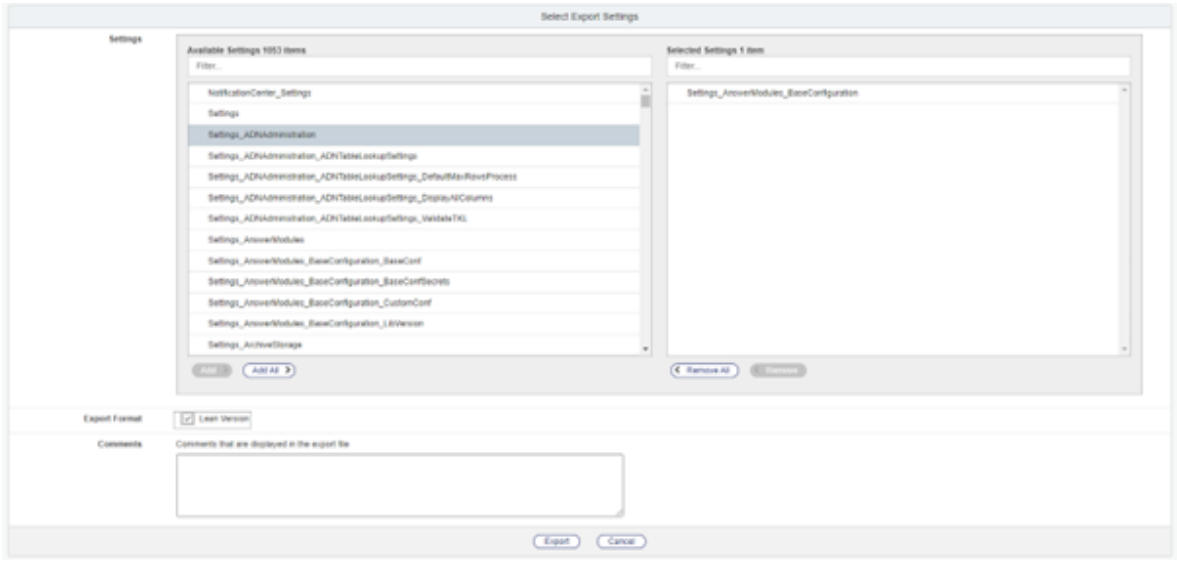
```
centos:/opt/am/sc/runtime/bin$ ./app.sh
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
```

```
AnswerModules Content Script Console
 type "help" for inline support
System:TEST>system -s LOCAL
Switched active system to 'LOCAL'
System:LOCAL>
```

The active system indicator in the command prompt should now indicate LOCAL.

- Synchronize ModuleSuite configuration parameters from the LOCAL system using the loadConfig command. To do this, you must first export the entire Module Suite configuration from the Content Server instance by following these steps:

  - ☑ Go to the Administration page of the Content Server and under ***Core System - Feature Configuration*** click on ***Import and Export Administration Settings***
  - ☑ Open ***Export Administration Settings***
  - ☑ In this page, add the ***Setting_AnswerModules_BaseConfiguration*** to the ***Selected Settings*** list, flag ***Lean Versions*** (as showed in the screenshot below) and click on ***Export*** to export the XML file containing the Module Suite configuration.



  - ☑ Copy eh file generated at the previous step on the server where Script Console is installed under the path `%AM_CONSOLE_DATA%/runtime/bin`
  - ☑ Run the command `loadConfig` specifying the file to be imported (-f, --file)

- The configuration is complete. Try a simple ls command to test the console

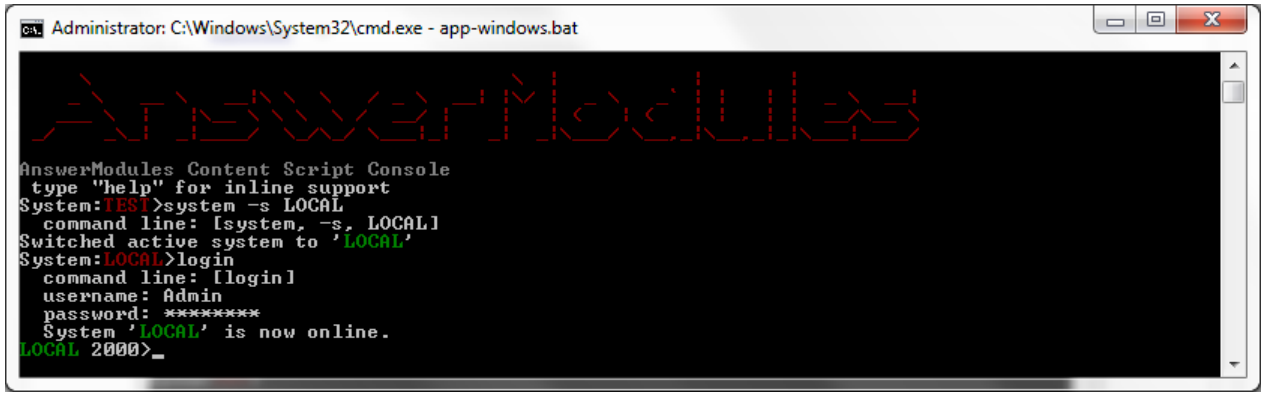
Connect to Content Server

Script Console does not require to be connected to a Content Server instance, in fact in most cases the two systems do not need to be connected. To execute actions and scripts against an active Content Server instance, you must log-in using valid user credentials.

Unix

```
System:LOCAL>login
  System 'LOCAL' is now online.
LOCAL 2000>loadConfig -m ALL

LOCAL 2000>
```

- Login to the LOCAL system using the `login` command



Unix

```
System:LOCAL>login
  System 'LOCAL' is now online.
LOCAL 2000>
```

The active system indicator in the command prompt should now turn green to indicate that the system is ONLINE

Installing Module Suite Extension Packages¶

Installation procedure¶

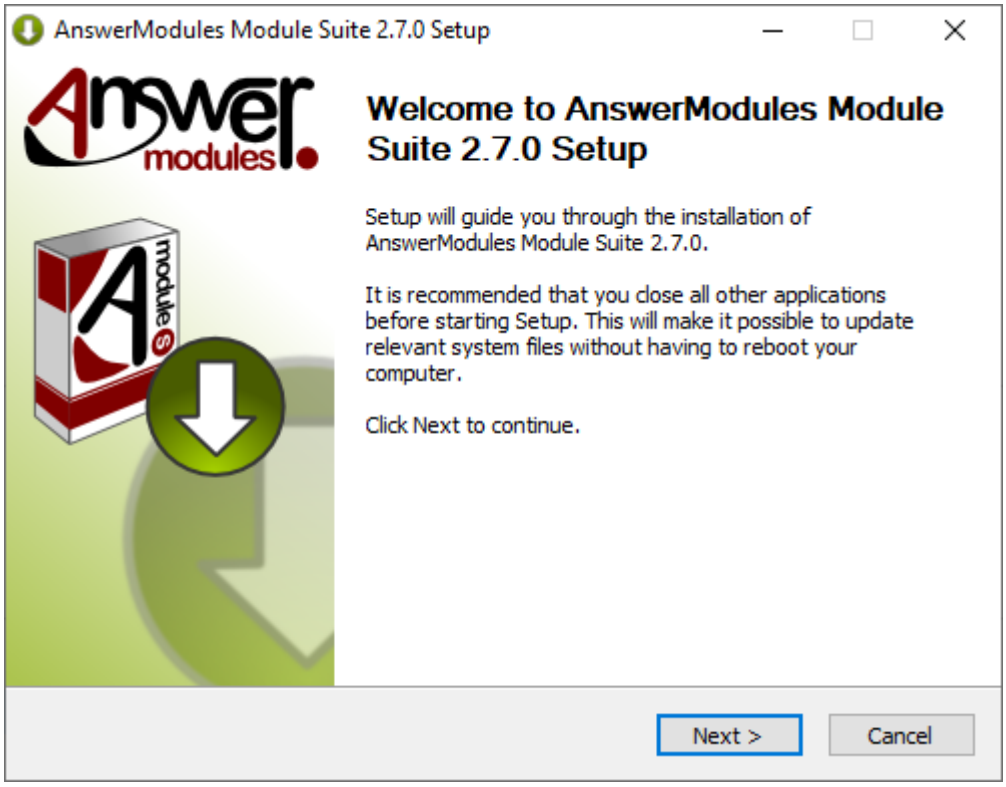
We will refer to the Content Server installation directory as **%OTCS_HOME%**

- Run the **Module Suite Content Script Master Installer** and install the desired extension packages.

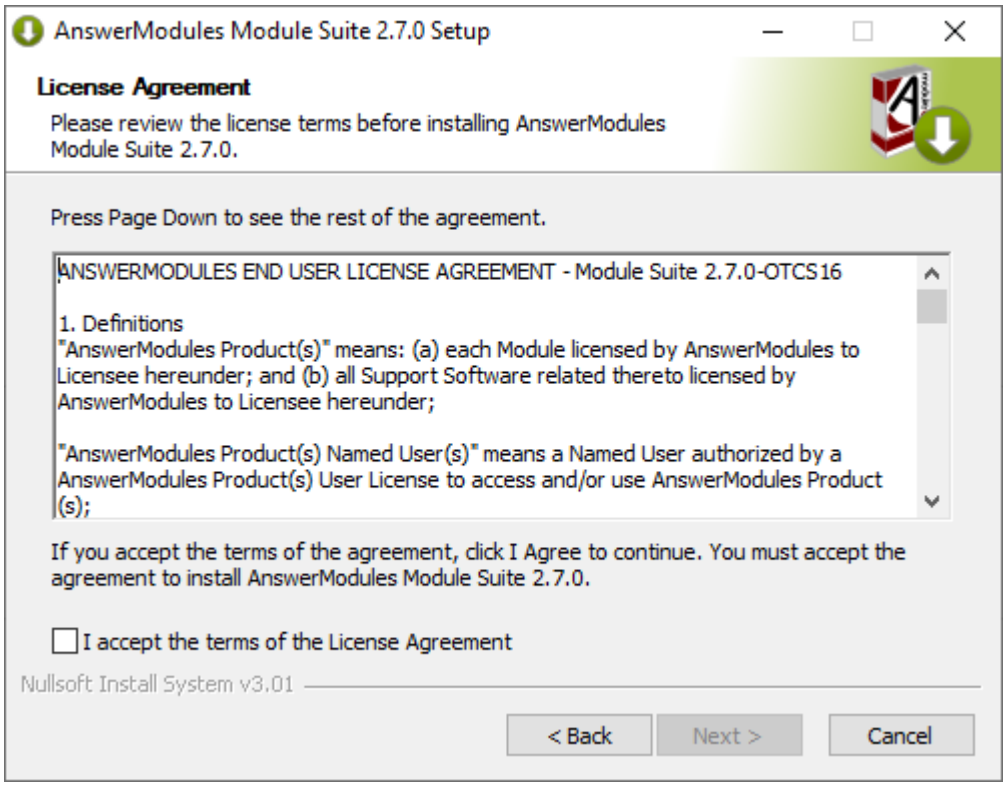
Step-by-step procedure

The following screens will guide you through the Content Script Module Master Installer steps required to install optional extension packages:

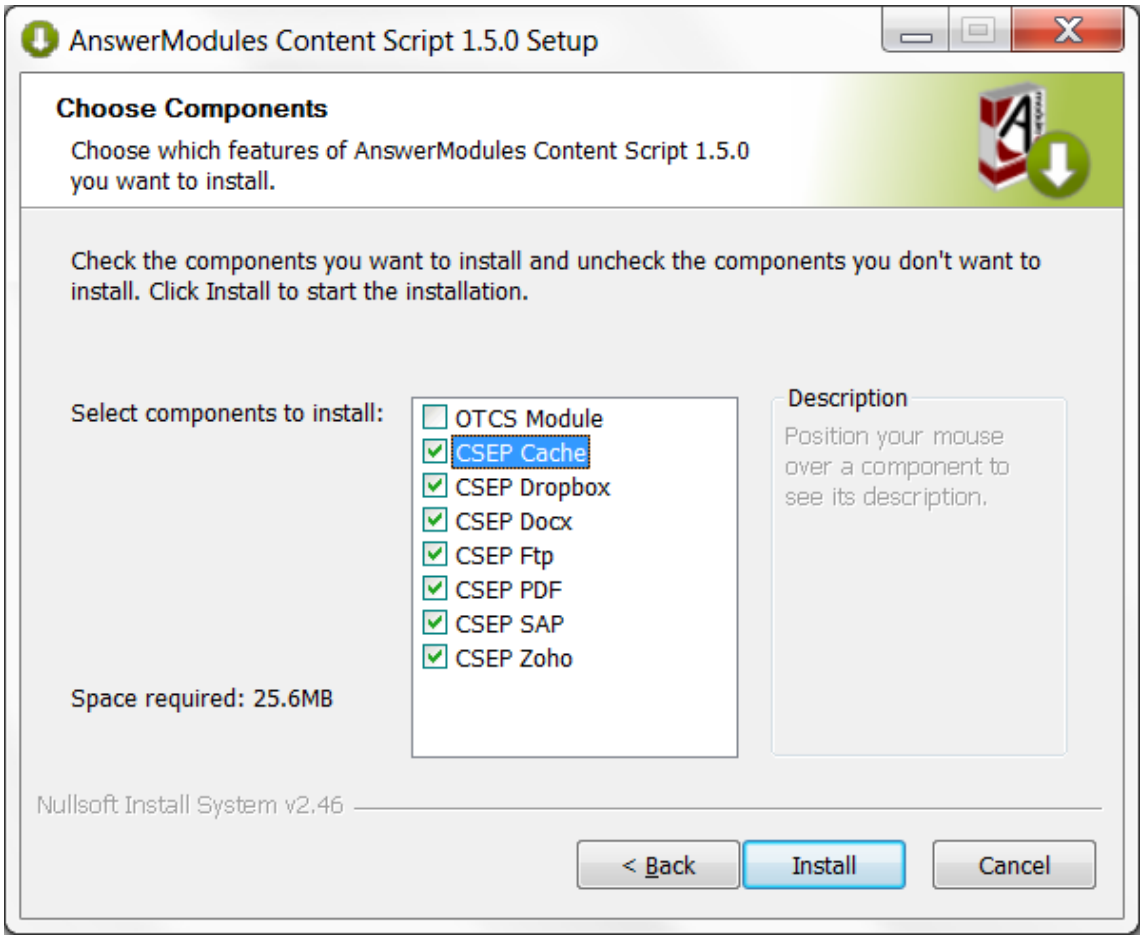
1. Welcome Screen: Select “Next” when ready to start the installation.



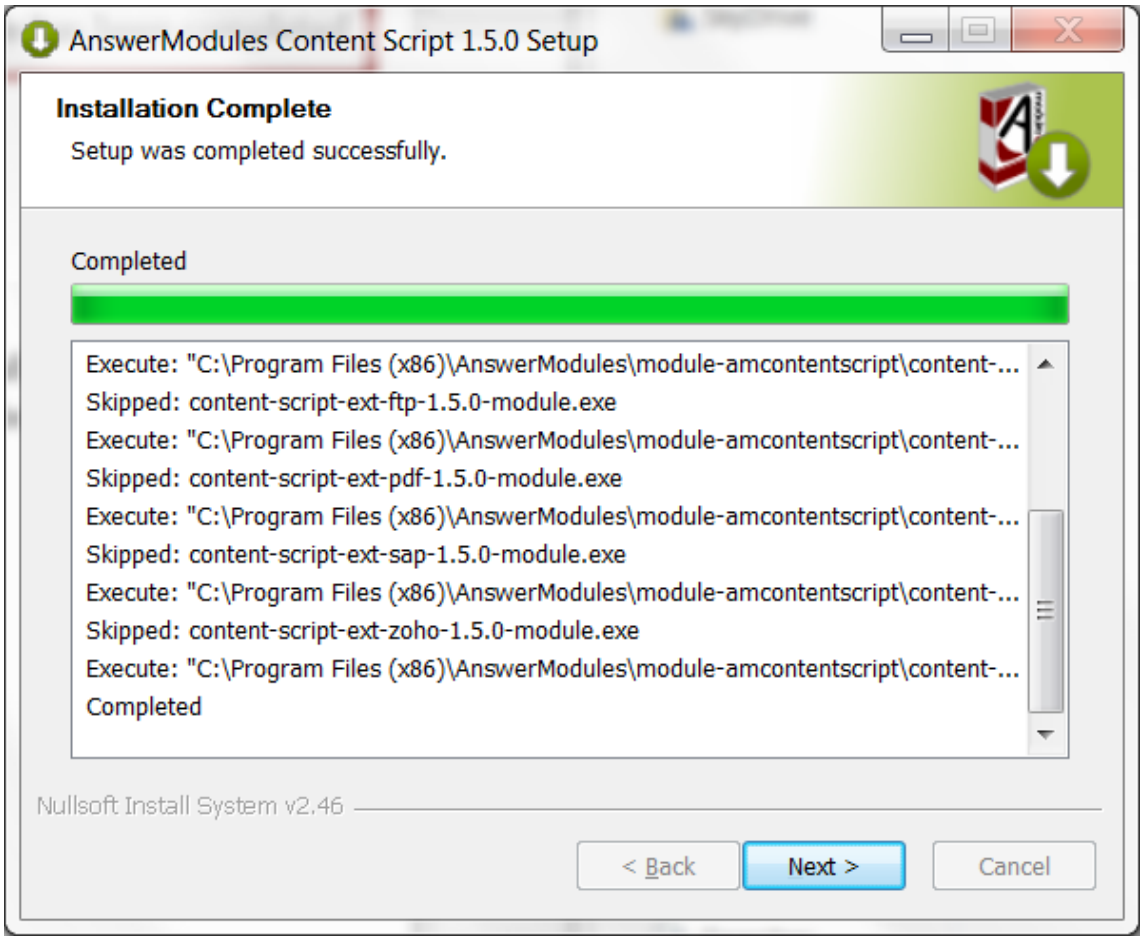
2. EULA Screen: Acceptance of the end-user license agreement is mandatory to proceed with the installation
   A copy of the agreement will be available, upon installation, in:
   %OTCS_HOME%/module/amcontentscript_X_Y_Z/license/EULA Accepting the End User Agreement is mandatory to proceed with the installation.
   Select “Next” when ready.



3. Components selection:Unselect the *OTCS Module* component. Select all of the extension components that are to be installed
   Select “Install” when ready.



4. Installation: The extension packages are automatically installed.
   Select “Next” when the procedure is complete.



Configure the Extension Packages

If you are installing extension packages on an already installed and properly configured Module Suite instance you have to update the module's **Base Configuration** following the procedure below:

- Stop and Start Content Server service to let the system load the newly installed Extension Packages
- Login as Administrator and access the Module administration panel
- From the Administration Home, select **AnswerModules Administration > Base Configuration**
- If necessary, change the core configuration or the configuration of the extension modules.
- Save the Base Configuration (even in case no changes were applied), and restart the OTCS services if prompted.

Since Module Suite version 3.2.0, updating the Base Configuration settings will only require a service restart for a limited number of options.

This is clearly marked in the Base Configuration UI and/or the documentation specific for the configuration setting.

Please note that whenever a restart is required as a consequence of the config change, the system will prompt to do so.

Rendition Extension Package¶

What is it?¶

The rendition extension package allows you to programmatically invoke a third party rendition engine to convert documents from one format to another, the most common use case is to convert HTML documents to PDF documents. Using the rendition extension package, you will be able to convert documents in real time and without interrupting the script execution flow.

The installation procedure for the rendition extension package isn't different from any other extension package, although it requires a couple of additional steps to be completed.

Install the third party rendition engine¶

The CS Rendition Extension package only provides the API to interface with a third party engine capable of converting documents.

This software is distributed separately by the third party and has to installed separately.

Although potentially compatible with different engines, the rendition extension package is pre-configured and tested to use on of the following options:

- an open engine AnswerModules R&D Team derived from the open source project Puppeteer *(https://github.com/puppeteer/puppeteer)* named **rend**
- an open source engine named wkhtmltopdf *(https://wkhtmltopdf.org/)* (deprecated)

The installation and configuration of the two above mentioned solutions is pretty similar.

rend¶

Installation (Windows)¶

External conversion engine package is provided as a compressed archive **rend-win.zip**. The Archive contains following items:

- ***chromium*** – folder containing an up to date version of Chromium *(https://www.chromium.org/Home)* engine.
- **rend** – pre-built NodeJS application leveraging Puppeteer *(https://github.com/puppeteer/puppeteer)*

To install it:

- [x] Extract the conversion engine package in the following location:

```
<OTCS_HOME>/module/anscontentscript_x_x_x/amlib/rend/dropin
```

Installation (Unix)¶

External conversion engine package is provided as a compressed archive **rend.tar.gz**. The Archive contains following items:

- ***chromium*** – folder containing an up to date version of Chromium *(https://www.chromium.org/Home)* engine.
- **rend** – pre-built NodeJS application leveraging Puppeteer *(https://github.com/puppeteer/puppeteer)*
- **run_rend** – a script that will be called by the Content Suite and will launch the application

To install it:

- ☑ Extract the conversion engine package in the following location:

```
<OTCS_HOME>/module/anscontentscript_x_x_x/amlib/rend/dropin

e.g.
>tar -C <OTHOME>/module/anscontentscript_x_y_0/amlib/rend/dropin -xvf rend.tar.gz
```

Note: files inside dropin folder should belong to user that is used to run Content Server service. Thus you can either perform extraction under the OTCS service user or change ownership of the extracted files accordingly.

Configuration¶

- ☑ Configure the Rendition extension package in order to use the ***rend*** executable in the Module Suite Base Configuration *(/administration/modulesuite/#base-configuration)*

  Section ***rend***

  Windows

Configuration Property	Configuration Property Value
amcs.rend.html2pdf.dropin	rend-win
amcs.rend.html2pdf.cmdline	"${source}" --cookie "${cookie}" -p "${destination}" --format A4 --marginBottom 100px --marginTop 120px --marginLeft 30px --marginRight 30px --scale 0.8 --viewport 1240x1754
amcs.rend.html2pdf.timeout	60000

  Unix

Configuration Property	Configuration Property Value
amcs.rend.html2pdf.dropin	run_rend
amcs.rend.html2pdf.cmdline	"${source}" --cookie "${cookie}" -p "${destination}" --format A4 --marginBottom 100px --marginTop 120px --marginLeft 30px --marginRight 30px --scale 0.8 --viewport 1240x1754
amcs.rend.html2pdf.timeout	60000
Configuration Property	**Configuration Proerty Meaning**
amcs.rend.html2pdf.dropin	The relative path to the engine's executable. For security reasons, the root of this path is the extension package's dropin folder.
amcs.rend.html2pdf.cmdline	The template of the command line instruction to be used when performing rendition ***(**)***. A few replacement tags can be used in this command line template. (a) ${source} : represent the absolute path for the input resource you want to render. Its value is automatically injected by the rendition extension package. Since the rendition extension package works on Content Script Resources, you do not have to worry about file system housekeeping. (b) ${destination} :represent the absolute path for the output resource, the engine is going to generate. Its value is automatically injected by the rendition extension package. Since the rendition extension package works on Content Script Resources, you do not have to worry about file system housekeeping. c ${cookie} : represent a local authentication cookie
amcs.rend.html2pdf.timeout	the default maximum wait time, in milliseconds, after which a rendition attempt will be aborted.

Dropin options

-“${source}” – replacement tag that will be substituted by the URL to the generated HTML Form. This argument is mandatory and not editable.

-ck, --cookie [cookie] – value will be replaced by replacement tag that corresponds to the current user’s session cookie. Should be in form “Name Value”. This argument is mandatory and not editable.

-p, --path \<path> – identifies target PDF file location. Value will be substituted by the replacement tag. This argument is mandatory and not editable.

-f, --format [format] – PDF option. Paper format. If set, takes priority over width or height options. Defaults to 'Letter'. Available options: Letter, Legal, Tabloid, Ledger, A[0-6].

-d – Debug is on. If specified debugging information is written to the log file. Use only for debugging purposes. Log file located in \<OTHOME>\logs\cs_rend.log or when running application manually in \<appDir>\log\cs_rend.log

-mb, --marginBottom [margin] - Bottom margin, accepts values labeled with units.

-mt, --marginTop [margin] - Top margin, accepts values labeled with units.

-mr, --marginRight [margin] - Right margin, accepts values labeled with units.

-ml, --marginLeft [margin] - Left margin, accepts values labeled with units.

-vp, --viewport [cookie] - PDF option. Set the viewport. Width and height of the page in pixels

-prt, --printmediatype - Use print media type. Boolean. Default: true.

-s, --scale [scale] - Scale of the webpage rendering.

-dhf, --displayHeaderFooter - Display header and footer. Boolean. Default: false.

-ht, --headerTemplate [template] - HTML template for the print header.

-ft, --footerTemplate [template] - HTML template for the print footer.

-pb, --printBackground - Print background graphics. Boolean. Default: true.

-pr, --pageRanges - Paper ranges to print, e.g., '1-5, 8, 11-13'. Defaults to the empty string, which means print all pages.

-w, --width [width] - Paper width, accepts values labeled with units.

-h, --height [height] - Paper height, accepts values labeled with units.

-wu, --waitUntil [choice] - WaitUntil accepts choices load, domcontentloaded, networkidle0, networkidle2. Defaults to 'networkidle2'.

For more detailed description of the option please refer to official Puppeteer documentation *(https://pptr.dev/#?product=Puppeteer&version=v3.0.1&show=api-class-page)*

wkhtmltopdf (Deprecated)¶

Deprecation Notice: wkhtmltopdf

The usage of `wkhtmltopdf` has been deprecated in Module Suite 3.0.0.

Recommended Migration

We strongly encourage customers to migrate to the `rend` package for PDF rendering.

Installation¶

- [x] Follow the software developers instructions to perform the installation on each server in the OTCS cluster on which the extension is needed.

- [x] Upon a successful installation, the main executable has to be made available to the Content Script Extension Package as a dropin.

  To do so:

  - locate the wkhtmltopdf installation path
  - locate the wkhtmltopdf.exe executable in the folder
  - copy the wkhtmltopdf.exe in the CS Rendition Extension package dropin folder, located in:

    ```
    <OTCS_HOME>/module/anscontentscript_x_x_x/amlib/rend/dropin
    ```

Configuration¶

- [x] Configure the Rendition extension package in order to use the ***wkhtmltopdf*** executable in the Module Suite Base Configuration *(/administration/modulesuite/#base-configuration)*

  Section ***rend***

Configuration Property	Configuration Property Value
amcs.rend.html2pdf.dropin	wkhtmltopdf
amcs.rend.html2pdf.cmdline	-B 10 -T 10 -L 5 -R 5 --viewport-size 1920x1080 ${source} --print-media-type --cookie ${cookie} --run-script "am_printFix()" ${destination}
amcs.rend.html2pdf.timeout	60000
Configuration Property	**Configuration Proerty Meaning**
amcs.rend.html2pdf.dropin	The relative path to the engine's executable. For security reasons, the root of this path is the extension package's dropin folder.
amcs.rend.html2pdf.cmdline	The template of the command line instruction to be used when performing rendition ***(**)***. A few replacement tags can be used in this command line template. (a) ${source} : represent the absolute path for the input resource you want to render. Its value is automatically injected by the rendition extension package. Since the rendition extension package works on Content Script Resources, you do not have to worry about file system housekeeping. (b) ${destination} :represent the absolute path for the output resource, the engine is going to generate. Its value is automatically injected by the rendition extension package. Since the rendition

Configuration Property	Configuration Proerty Meaning
	extension package works on Content Script Resources, you do not have to worry about file system housekeeping. c ${cookie} : represent a local authentication cookie
amcs.rend.html2pdf.timeout	the default maximum wait time, in milliseconds, after which a rendition attempt will be aborted.

(**)

Please refer to the third-party rendition engine's guide for a detailed explanation of all the available command line parameters

Content Script Extension for SAP¶

What is it?¶

Content Script Extensions for SAP allows to integrate Content Script with the SAP™ ERP through RFCs (Remote Functions Calls).

The integration allows you to perform the following:

- connect to multiple SAP™ systems through JCo APIs;
- invoke standard and custom SAP™ functions for retrieving ERP's information;
- invoke standard and custom SAP functions for updating ERP's information;

SAP™ JCo Library Required

This extension package requires the SAP™ JCo library *(https://support.sap.com/en/product/connectors/JCo.html)* to be available in the extension repository `<OTHOME>/module/anscontentscript_x_y_z/amlib/sap` and is certified for use with SAP™ JCo version (3.0.6) when used on OpenText Extended ECM and version (3.0.10) when used on CSP. SAP™ JCo library *(https://support.sap.com/en/product/connectors/JCo.html)* can be downloaded from SAP™ website.

Extension setup¶

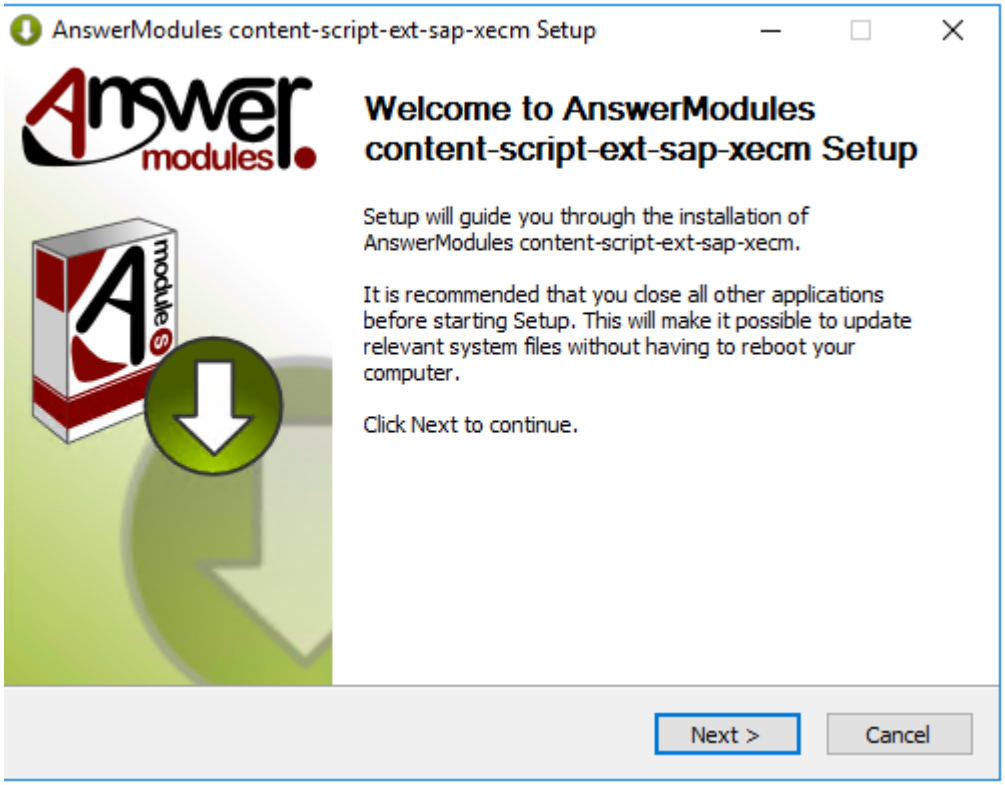
The Content Scripting extension for SAP is part of the Module Suite bundle.

Below is the step by step guide on how to install the Extensions for SAP. **Note**: For the general Module Suite and Module Suite Extensions Packages installation procedure please refer to "Installing the suite" *(/installation/installation/)* section

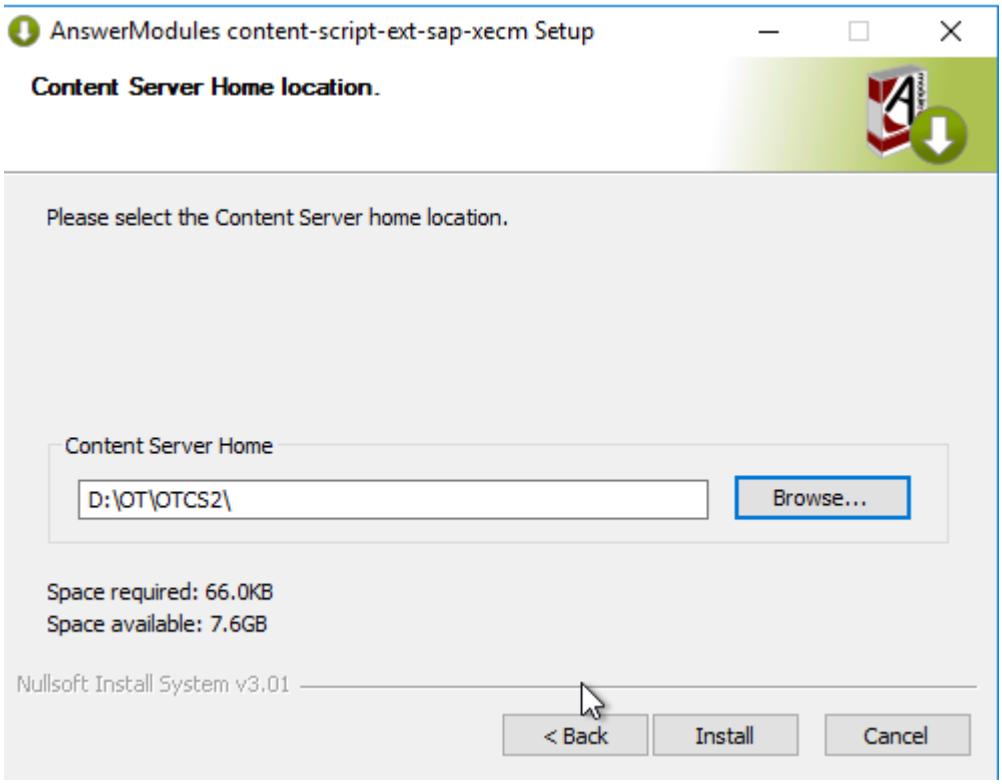
Installing the Content Script Extension for SAP¶

Run the Content Script SAP Extension installer and follow the installation wizard steps:

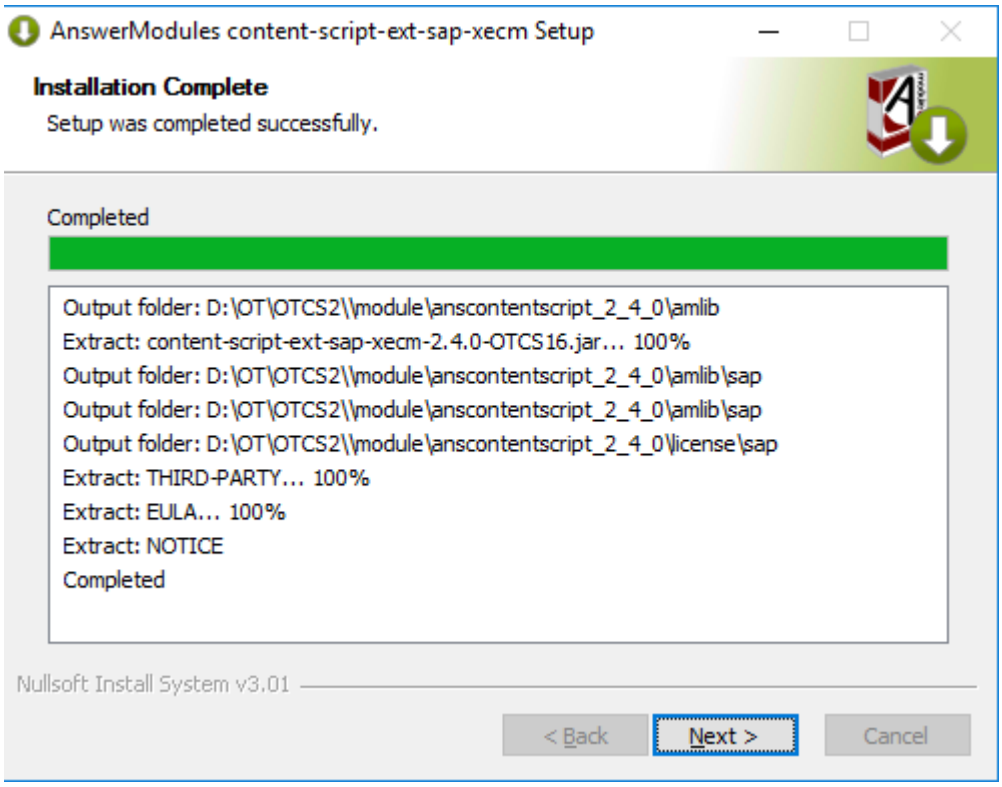
- Select "Next" when ready to start the installation.



- Accept all the required license agreements
- The installer will prompt you for the location of the installed Content Server. Browse to your **OTCS_HOME** and select "Next".



- [x] Click "Install" to start the installation
- [x] The installation of the required libraries will be performed



- [x] Deploy SSAP™ JCo in the extension package repository: `<OTCS_HOME>/module/anscontentscript_2_x_0/amlib/sap`. The Content Script extension for SAP relies on SAP Java Connector (SAP JCo) to support outbound communication with the SAP Server. SAP JCo

relies on a native bridge to implement the communication with the SAP server. This native bridge is implemented by the SAP JCo native library (sapJCo.dll). Both the SapJCo jar file and dll must be copied in the extension package repository.

To deploy SapJCo library follow this simple procedure:

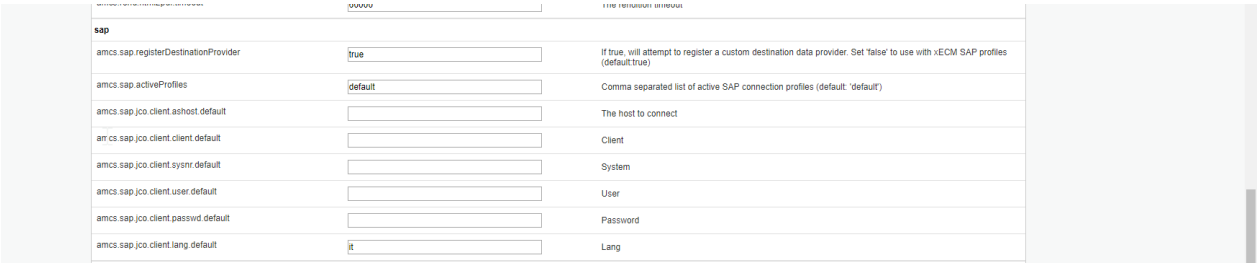
- Stop Content Server service
- Copy library files to the destination mentioned above
- Start Content Server service

Deploy on clustered environment

In case of a clustered Content Server installation the above steps should be performed on every cluster node.

Installation validation¶

If the Content Script Extension for SAP has been successfully installed, a new configuration section should appear in the Base Configuration *(/administration/modulesuite/#base-configuration)* page:



Configuration options¶

List of available parameters specified below:

Configuration Property	Configuration Property Meaning
amcs.sap.registerDestinationProvider	Determines whether the existing xECM connection or a custom connection should be used.When set to TRUE the custom destination data provider is used; when set to FALSE the existing configured SAP xECM connection is used.
amcs.sap.activeProfiles	List of the currently active and configured sap extension profiles. As many other extension packages Content Script Extension for SAP allows you to define multiple configuration profiles in order to manage multiple connections towards different systems.

Configuration Property	Configuration Property Meaning
amcs.sap.JCo.client.ashost.default	Target SAP System server hostname
amcs.sap.JCo.client.client.default	Target SAP System Client number
amcs.sap.JCo.client.sysnr.default	Target SAP System ID
amcs.sap.JCo.client.user.default	Target SAP System username to logon with
amcs.sap.JCo.client.passwd.default	Target SAP System password for the specified username
amcs.sap.JCo.client.lang.default	Language to use for the connection

OpenText Activator

If you have not installed the "OpenText Activator for SAP Solutions" module on your system, you can only use the custom destinations. In this case it is necessary to install the SAP JCo version compatible with your environment.

Installing Extension for DocuSign

Prerequisites¶

This guides assumes the following components to be already installed and configured:

- AnswerModules ModuleSuite
- Script Console (*OPTIONAL - only for DocuSign webhook configuration*)

The following information will be required to complete the configuration procedure:

- DocuSign API key
- Docusign API credentials

Authentication Options

The Content Script extension supports two different authentication options when invoking DocuSign APIs:

- Username / Password
- Account GUID / RSA Certificate

Refer to the official DocuSign REST API guides *(https://developers.docusign.com/docs/esign-rest-api)* for details on how to generate your credentials.

We will refer to the Content Server installation directory as **OTCS_HOME**

We will refer to the Script Console installation directory as **SCRIPT_CONSOLE_HOME**

Installation procedure¶

The **Module Suite DocuSign Extension** includes two components:

- Content Script Extension for DocuSign

  *This component enables the **docusign** service API in Content Script. The service is the entry point to integrating DocuSign functionality within your applications.*

- Script Console Extension for DocuSign (*Optional*)

  *This component enables a **DocuSign webhook endpoint** on Script Console. It is only required if you want to receive automatic update notification from DocuSign whenever an envelope status changes. For more details, refer to the official DocuSign REST API Guides (https://developers.docusign.com/platform/webhooks/connect/create-webhook-listener/) related to this topic.*

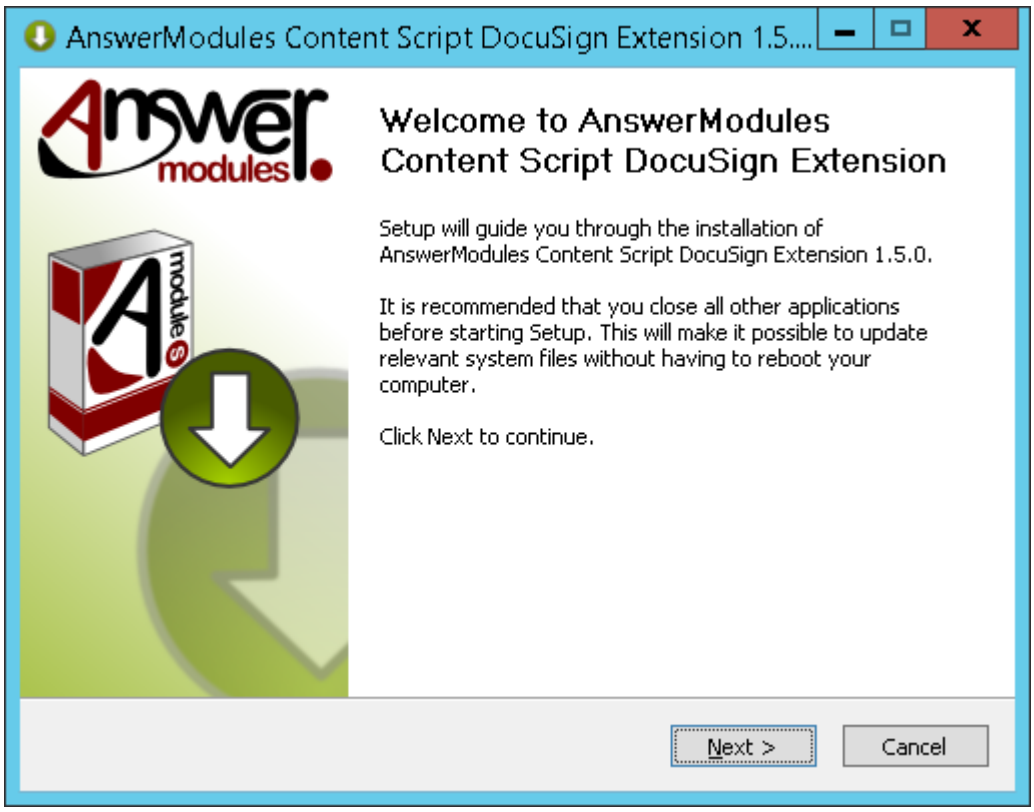
Installing the Content Script Extension for DocuSign¶

Run the Module Suite DocuSign installer:

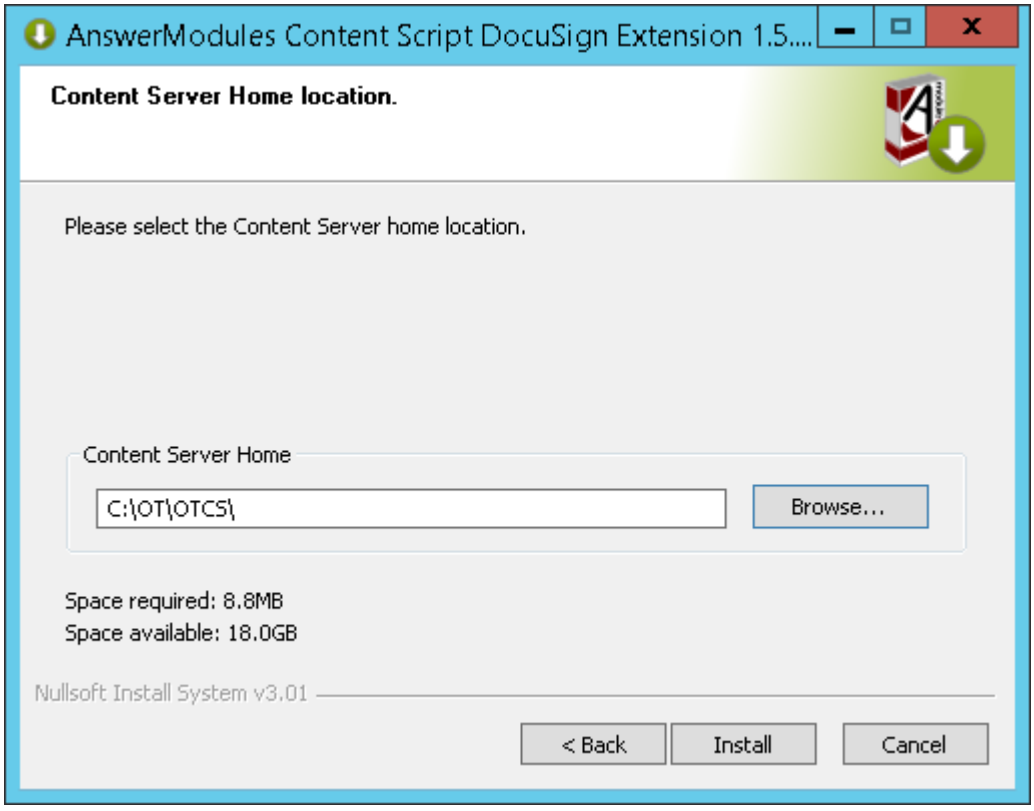
```
module-ansmodulesuitedocusign-1.5.0-OTCSxxx.exe
```

Follow the installation wizard steps:

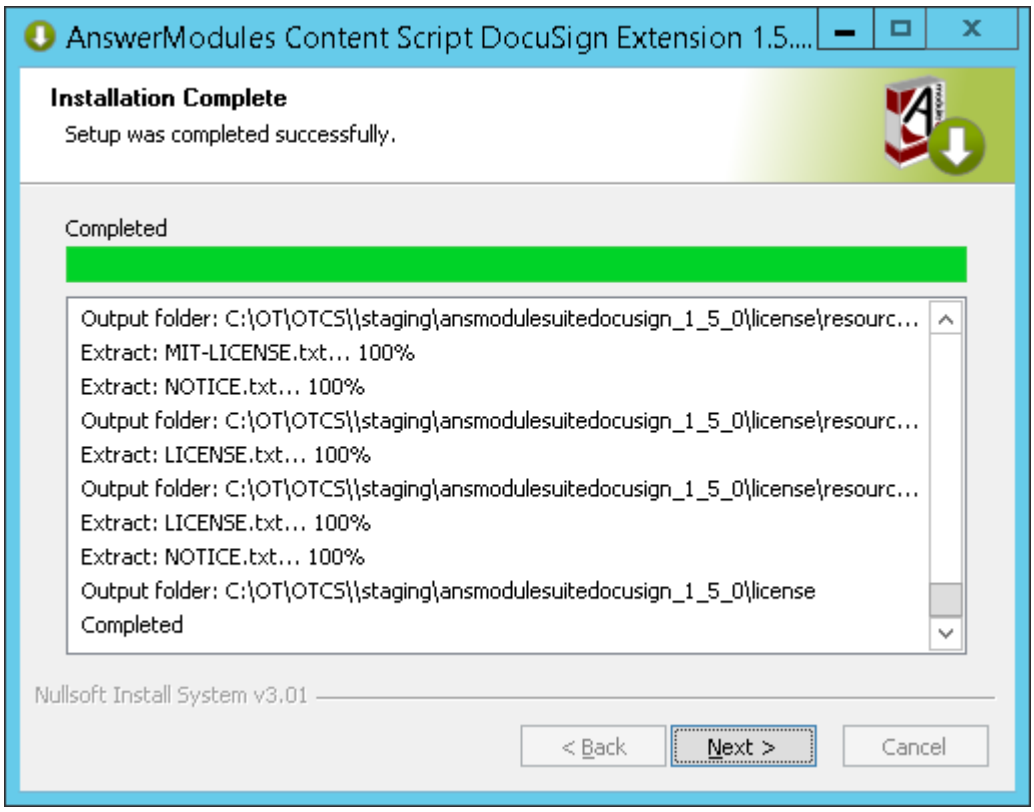
- [ ] Select "Next" when ready to start the installation.



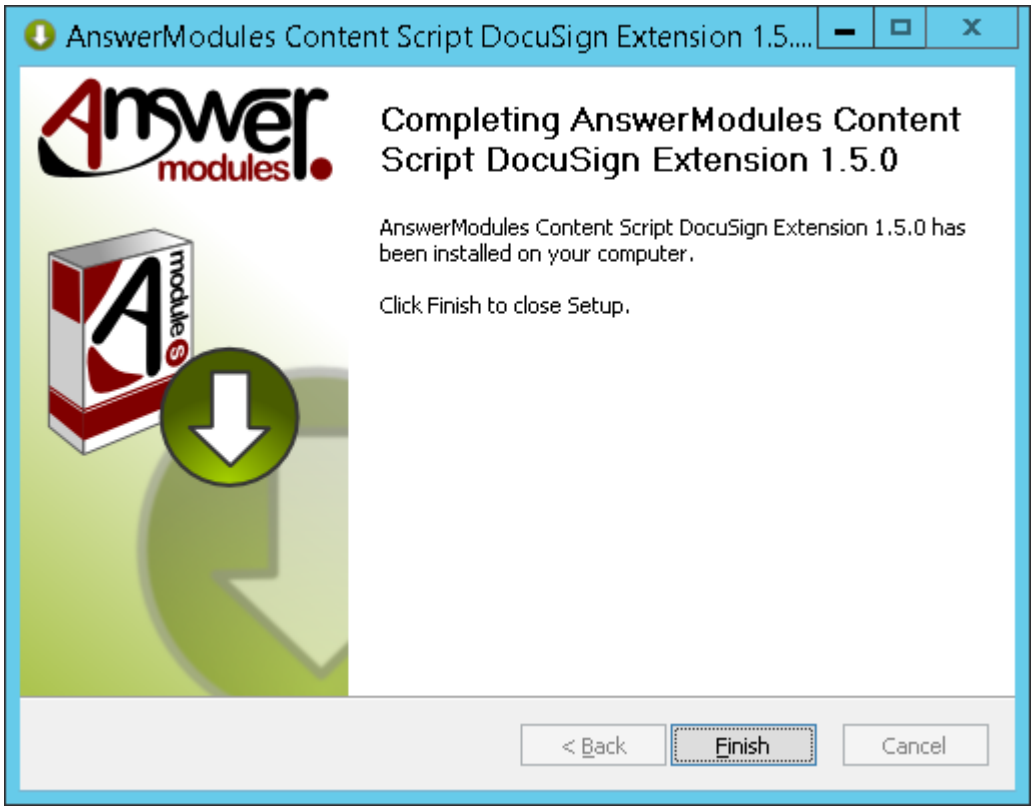
☐ The installer will prompt you for the location where Content Server is installed. Browse to your **OTCS_HOME** and select "Next".



☐ Review the installation steps for each component to be installed.



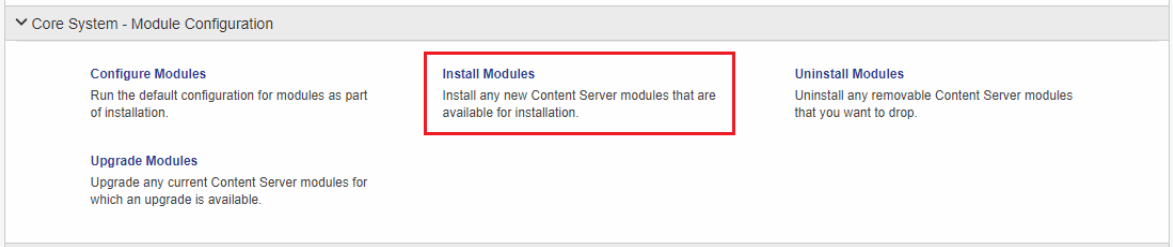
- [ ] Click "Finish" to complete the unpacking of the module



Staging

At this point, the Module has been deployed in the Content Server Staging folder and is available for module install through the Content Server administration pages.

- [ ] Access the Content Server Admin pages > **Core System - Module Configuration** > **Install Modules**



- [ ] Locate the **AnswerModules Module Suite extension for Smart UI** module and proceed with installation

- [ ] Restart the OTCS services when prompted in order for the installation to be completed.

Installing the Script Console Extension for DocuSign (OPTIONAL)¶

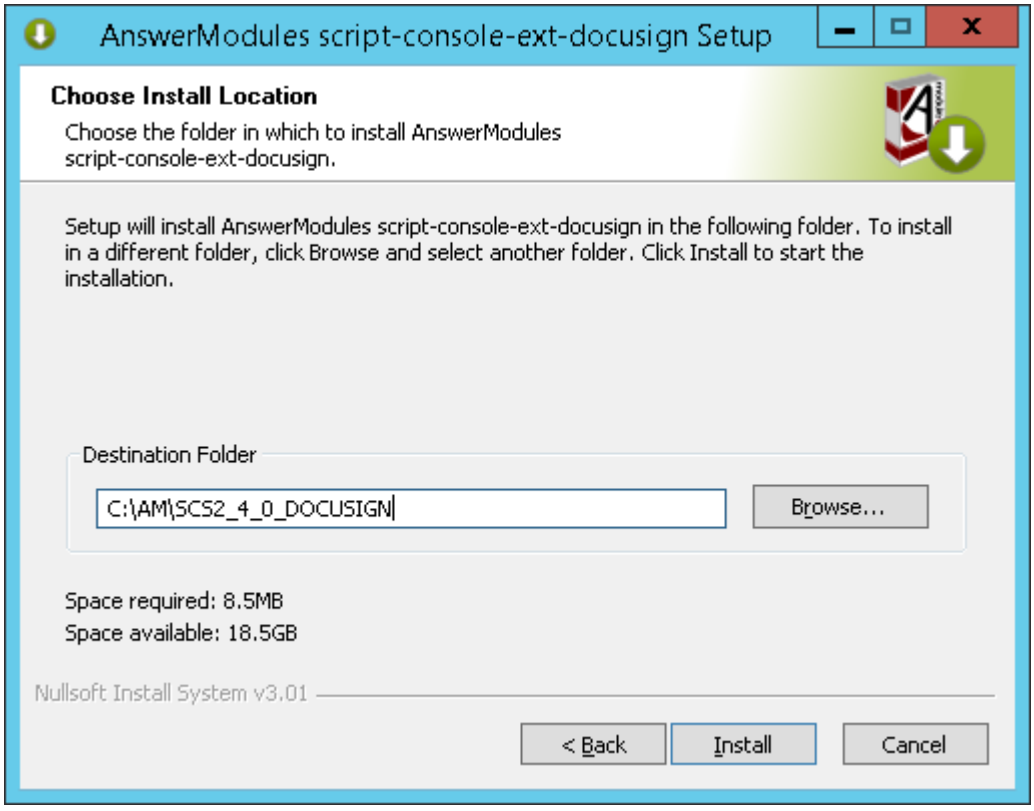
Run the Script Console DocuSign Extension installer:

```
script-console-ext-docusign-2.4.0-OTCSxxx.exe
```

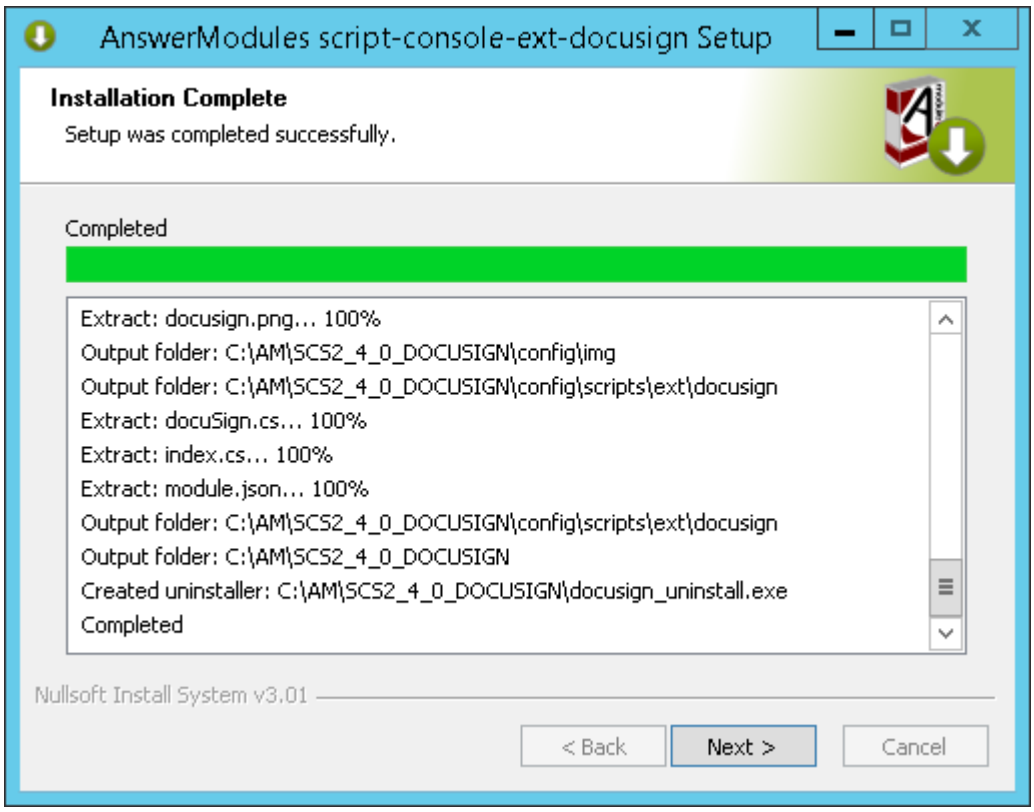
Follow the installation wizard steps

- [ ] Select "Next" when ready to start the installation.

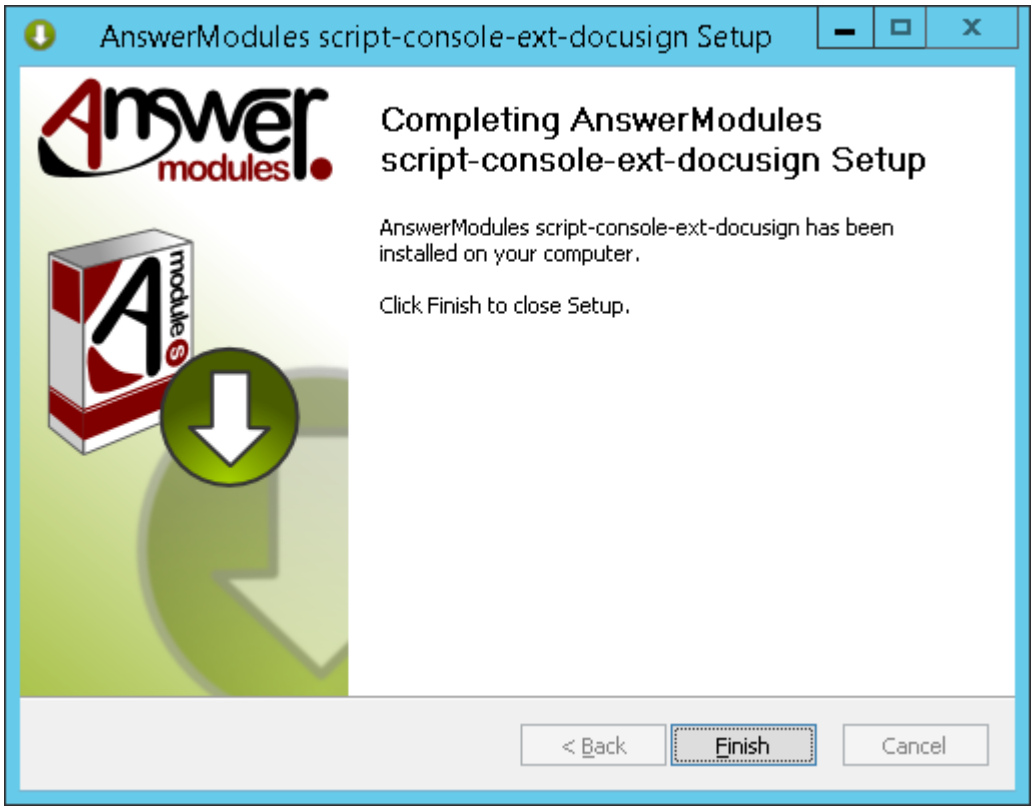
☐ The installer will prompt you for the location where your target Script Console instance is installed. Browse to your **SCRIPT_CONSOLE_HOME** and select "Next".



☐ Review the installation steps for each component to be installed.



- [ ] Click "Finish" to complete the installation



Update the security configuration to allow access to the webhook endpoint. Edit the Script Console security config file:

```
<SCRIPT_CONSOLE_HOME>\config\cs-console-security.xml
```

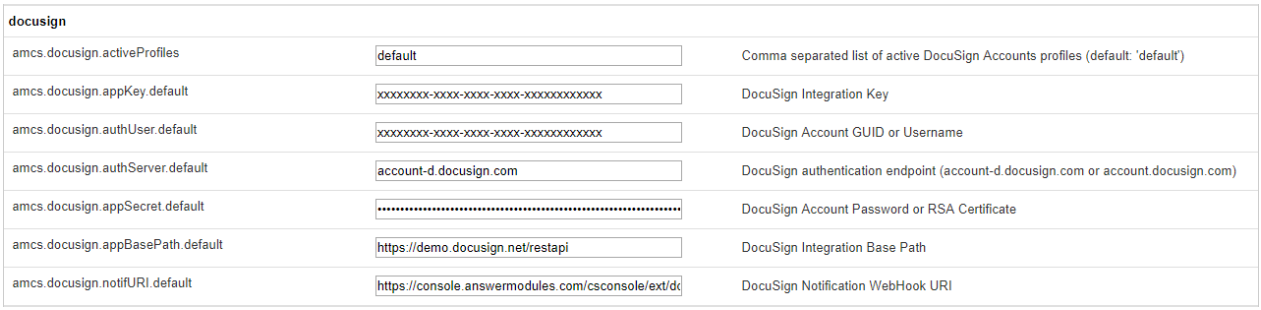
Add the following rule:

```
<s:http pattern="/ext/docusign/docuSign.cs" security="none"/>
```

Configuration¶

The DocuSign Connector requires a few configuration parameters in order to be able to communicate with DocuSign systems using the eSignature REST APIs.

In the OTCS Admin pages > AnswerModules Administration > Base Configuration section, complete the "docusign" API configuration.



The following parameters are available:

Key	Description
amcs.docusign.activeProfiles	Comma separated list of active DocuSign Accounts profiles (default: "*default*"). This is a local identifier and will not be sent over to DocuSign. It is only relevant when more than one set of configurations has to be specified.
amcs.docusign.appKey.default	DocuSign Integration Key: identifies your app for the DocuSign platform.
amcs.docusign.authUser.default	DocuSign Account GUID or Username
amcs.docusign.authServer.default	DocuSign authentication endpoint. This can be either **account-d.docusign.com** for sandbox testing or **account.docusign.com** for a production account.
amcs.docusign.appSecret.default	DocuSign Account Password or RSA Certificate. If an Account GUID has been provided in the "*amcs.docusign.authUser.default*" field, than this MUST be an RSA Certificate private key. Otherwise, if a

Key	Description
	Username has been provided, this MUST be the account password.
amcs.docusign.appBasePath.default	DocuSign Integration Base Path. This can be either https://demo.docusign.net/restapi *(https://demo.docusign.net/restapi)* for sandbox testing or https://www.docusign.net/restapi *(https://www.docusign.net/restapi)* for a production account.
amcs.docusign.notifURI.default	DocuSign Notification WebHook URI. This is the **absolute**, **publicly accessible URL** that DocuSign will call for push notifications. It refers to the endpoint installed on your Script Console instance. This value is **OPTIONAL** and only required if using the push notifications.

RSA Certificate format

If using the RSA certificate authentication (combined with an account GUID), the following requirements must be met:

- RSA Certificate must be stored on a single line.
- Line breaks must be replaced with line feeds (*\n*).
- The *"-----BEGIN RSA PRIVATE KEY-----"* block and *"-----END RSA PRIVATE KEY-----"* must be included.

Example:

```
-----BEGIN RSA PRIVATE KEY-----\nxxx....xxx\nxxx....xxx=\n-----END RSA PRIVATE KEY-----\n
```

Save the Base Configuration and restart Content Server services when requested

Admin dashboard¶

The **Module Suite DocuSign Extension** supports the storage of a local copy of the signing envelope details within Content Server. The envelope status can either be periodically updated through a scheduled job, or automatically updated using push notifications by DocuSign (using a webhook pattern). An overview of the status of current and past envelopes can be visualized using the DocuSign Connector Admin dashboard.

The dashboard is a Content Script based tool that can be installed in the Content Script Volume using the Module Suite import/upgrade tool.

Before running the import, you should make the lib file available to the tool with the following steps:

- ☐ On the server, navigate to the DocuSign Extension Module folder

```
<OTCS_HOME>\module\ansmodulesuitedocusign_1_5_0\library
```

and locate the file named **docusign integration.lib**.

- [ ] Copy the file to the **library** folder within the Content Script Module:

```
<OTCS_HOME>\module\anscontentscript_2_4_0\library
```

Now that the library is available, proceed to the import with the following steps:

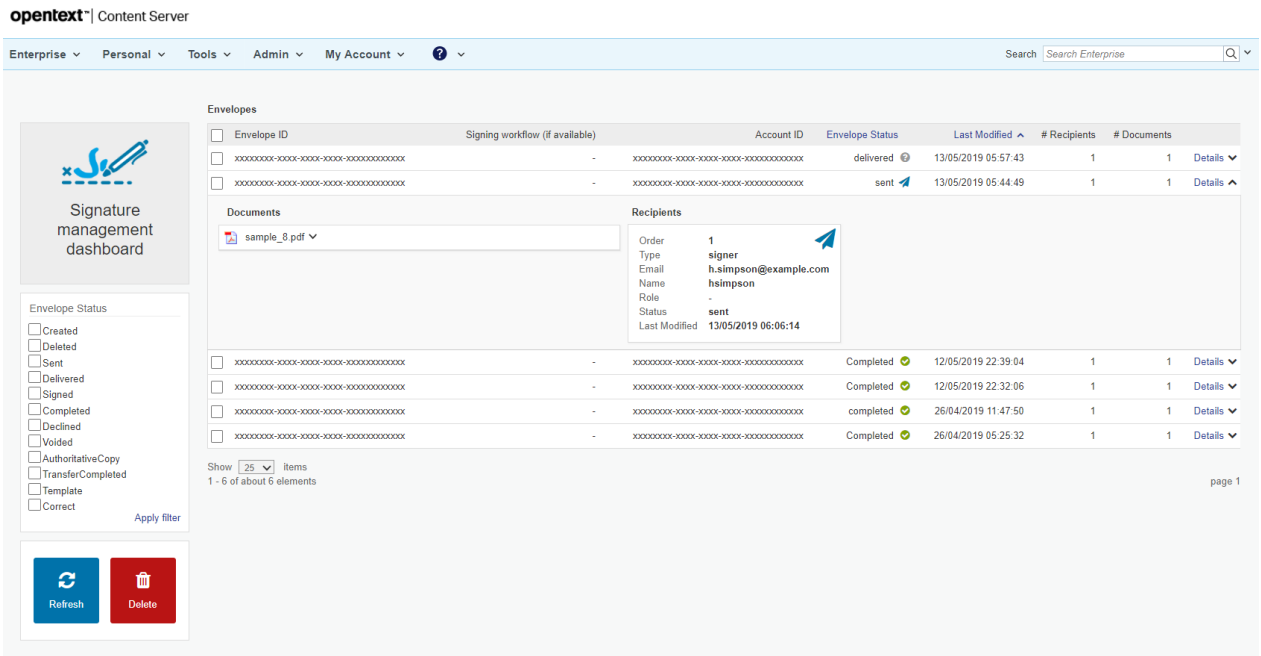
- [ ] In a web browser, open the Module Suite Administration Base Configuration page. If working in a clustered environment, make sure you connect to the same server on which the library file has been copied.

- [ ] Use the "*Import*" tool within the base configuration to import the **DocuSign Integration** library

Once the import is complete, you will be able to access the dashboard by navigating to the following Content Server location:

```
Content Script Volume > DocuSign Integration > CSTools
```

and running the **Dashboard** script.



Applying HotFixes

Module Suite hotfixes are typically distributed in the form of compressed file archives (.zip files).

The content of the archive is a folder structure that mirrors the structure of the Content Server installation directory (e.g. "E:\Opentext" or "/opt/opentext/otcs").

Below an exemplar structure of an "hotfix" archive:

```
├ module
|    ├ anscontentscript\_X\_Y\_Z
|    ├ amlib
|    ├ ...
|    ├ ...
|    ├ hotfixes
|    |  └──── hotFix_ANS_XYZ_###.hfx
|    ├ ...
|    ├ ojlib
|    └ ...
|
└ support
     ├ anscontentscript
     ├ ...
     ├ amui
     |  └──── js
     |        └────...
     └ ...
```

Naming convention

AnswerModules hotfixes follow a simple naming convention: they are all preceded by **hotFix_ANS_** followed by an optional string that identifies the AnswerModules product (e.g. **DS** for DocuSign Connector) (if absent the hotfix must be consider for Module Suite) followed by three digits identifying the version of the AnswerModules product followed by three digits identifying the hotfix followed by an optional string that identifies the OpenText Content Suite version the hotfix is compatible with.

e.g.

hotFix_ANS_240_001.zip
: Hotfix 001 for Module Suite version 2.4.0

hotFix_ANS_DS_150_002_CS16.zip
: Hotfix 002 for DocuSign Connector version 1.5.0 to be utilized on Content Server version 16.0.X

hotFix_ANS_SMUIEXT_150_001.zip
: Hotfix 001 for AnswerModules Smart View extension version 1.5.0

cumulative_hotFix_ANS_240_CS16X_009_024
: Cumulative hotfix (containing hotfixes from 009 to 024) for Module Suite version 2.4.0 to be utilized on Content Server 16.0.X

Hotfixes deployment¶

To install an hotfix the files provided in the hotfix archive must be deployed within the Content Server installation directory in order to overwrite existing files and/or to add new files to the AnswerModules product binaries.

The suggested procedure for installing an hotfix is the following:

- ☑ Extract the archive in a temporary folder;
- ☑ Read the patch installation notes carefully. The installation notes come in the form of a text file ending with **.hfx** located within the ***module/anscontentscript_x_y_z/hotfixes*** folder. The installation notes contains information about the issues addressed by the hotfix and any additional deployment instructions to follow;

cumulative hotfix

In case of a cumulative hotfix, carefully read all the hotfixes installation notes.

- ☑ Check the contents of the archive and backup all files in installation folder of the Content Server that will be overwritten by the hotfix;

Unless otherwise instructed by the hotfix installation notes:

- ☑ Stop the Content Server services;
- ☑ Copy the contents of the hotfix in the Content Server installation directory or follow hotfix's more specific instructions for deployment;
- ☑ Restart the Content Server services

Important notes

- Always read the hotfix notes before deploying the hotfix. Some hotfixes require additional operations to be performed before or after deploying the binaries;
- Always perform a backup of the patched binaries;
- Make sure that the version of the hotfix matches exactly the version of the target AnswerModules product and OpenText Content Suite environment.
- Hotfixes are identified by a progressive numbering. It is imperative that hotfixes are deployed respecting the **correct sequential order**, as it is possible that the same resources are patched by different hotfixes (e.g. ***hotFix_ANS_260_002.zip*** (progressive number: 2) **must not** be installed after ***hotFix_ANS_260_003.zip*** (progressive number: 3). If, for any reason, an hotfix has been skipped and has to be later installed on a system, all subsequent hotfixes **must be reinstalled** in order to ensure that no newer change has been reverted
- When OpenText Content Suite is running on a clustered environment, hotfixes must be installed on all the servers on which Content Suite is deployed.

uninstallation

Uninstalling Module Suite

Uninstallation procedure¶

We will refer to the Content Server installation directory as **%OTCS_HOME%**

Before proceeding with the uninstallation of Module Suite modules you need to complete some housekeeping routines. These routines are not strictly mandatory and should only be performed if you do not intend to reinstall the Module Suite on your system in the future.

- [ ] Shutdown CSEvents feature:

  This feature generates records in the Distributed Agent framework table, which are then managed by the **CallbacksManagerCS** handler. After uninstalling the Content Script module this type of handler will not be longer available, with the result that several errors will be generated in the DA framework's tables. To prevent these errors from occurring, it is safer to disable the feature completely and wait for all occurrences of this type of activity to be processed by the DA.

  - From the Administration Home, select **AnswerModules Administration > Base Configuration**, then enter **34** in the **amcs.core.debugEnabled** property and save the current configuration.

> **RESTART REQUIRED**
>
> A service rest of all the nodes that are part of your cluster is required.

> **amcs.core.debugEnabled is now 'Module Suite - Configuration Options'**
>
> In recent version of Module Suite the property **amcs.core.debugEnabled** has been associated with the label **Module Suite - Configuration Options** in the Base Configuration

  - Once all the nodes have been restarted wait until all the occurences of CallbacksManagerCS jobs have been processed and removed from the DA table. You can monitor this process by executing the query below:

```
select count(1) as "Total", 'WorkerQueue' as "Queue" from  WorkerQueue where Handle
union all
select count(1) as "Total", 'WorkerQueuePending' as "Queue" from  WorkerQueuePendin
union all
select count(1) as "Total", 'WorkerQueueCurrent' as "Queue" from  WorkerQueueCurren
```

- [ ] Delete all Content Server's columns or facets having a Content Script script as their datasource.

- [ ] Stop and delete all instances of worklows using Module Suite modules. Upon Module Suite uninstallation all the currently active workflows, which make use of a feature related to one of the Module Suite modules, will not be able to continue correctly, to avoid errors you must wait for these workflows to end or stop and delete them.

> **Modify Workflow Map**
>
> Remove any Content Script Step, Content Script Workpackage, Content Script Event Script from all your Workflow Maps

- [ ] Stop any scheduled script
    - From the Administration Home, select **AnswerModules Administration > Manage Content Script Scheduling** unschedule any previously scheduled Content Script script.
    - Wait the completion of any previously scheduled script execution. You can monitor this process by executing the query below:

```
select count(1) as "Total", 'WorkerQueue' as "Queue" from  WorkerQueue where Handle
union all
select count(1) as "Total", 'WorkerQueuePending' as "Queue" from  WorkerQueuePendin
union all
select count(1) as "Total", 'WorkerQueueCurrent' as "Queue" from  WorkerQueueCurren
```

- [ ] (OPTIONAL) Collect and delete all the Content Script, Smart Pages, and Beautiful WebForm Views Object objects on your system.
    - Although not strictly necessary, this action will prevent you from having objects on your system that the application can no longer handle correctly. In order to easily find collect and delete the afore mentioned objects we suggest you to create and execute the script below, which it will create in the same container where the script was created a collection containing all the scripts pages and views in your system.

```
collection = docman.createCollection(self.parent, "Module Suite Objects", "Module S
/*
43100 BWF Views
43200 Content Script
43300 SmartPages
*/
nodes = docman.getNodesFastWith(sql.runSQLFast("""select distinct DataID "DataID" f
collection.addNodes(nodes)
```

> **Execute the script as Admin**
>
> Don't forget to create and run the above script as an "Admin" user to make sure you can collect all objects on your system regardless of the associated permissions.

- [ ] (OPTIONAL) Delete the Content Script Volume and its content.
    - Although not strictly necessary, this action will prevent you from having objects on your system that the application can no longer handle correctly. From the Administration Home, select **AnswerModules Administration > Open The Content Script Volume** once in the volume delete the volume's content.
- [ ] Delete Beautiful WebForm SmartEditor table.
    - From the Administration Home, select **AnswerModules Administration > Base Configuration** then click on the link **DELETE** under the **Manage Beautiful WebForms database** section. The action will require confirmation.
- [ ] Using standard Content Server features uninstall all the Module Suite modules

Uninstallation complete

The Module Suite is no longer on your system. We miss you already.

configuration performances-tips productive

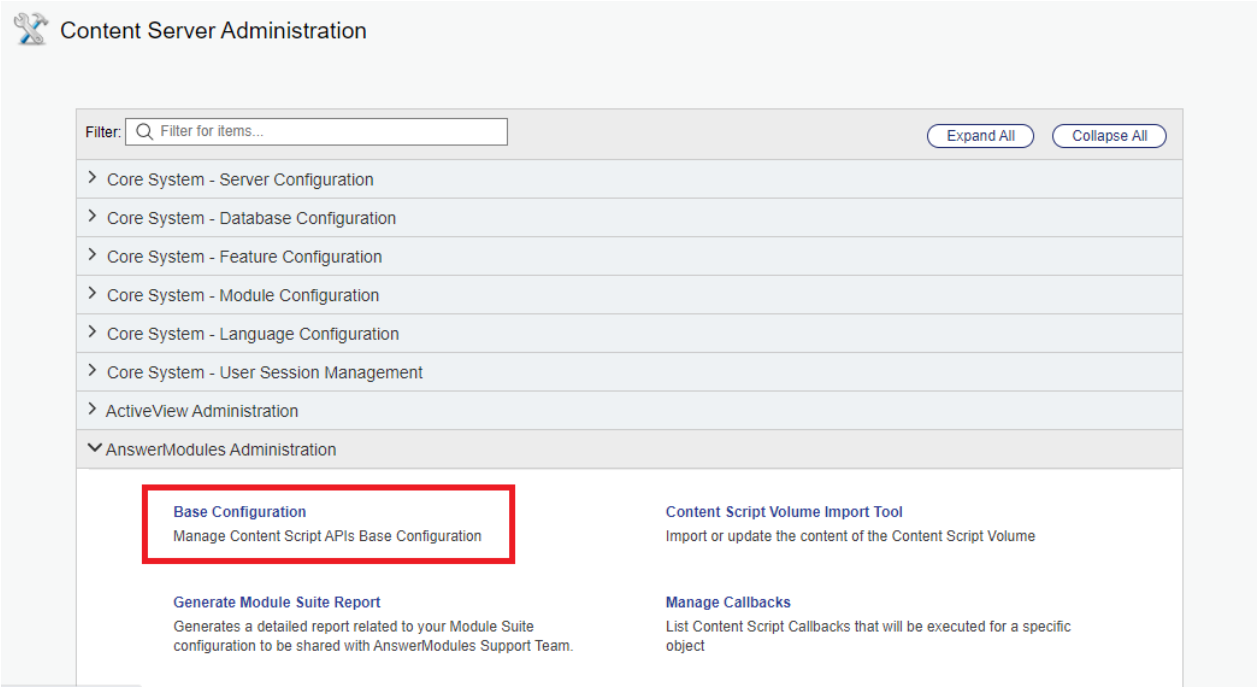
Introduction¶

In a production environment, certain conditions are often present that facilitate performance optimizations. Unlike in development or testing environments, configurations in production are expected to remain stable, with infrequent changes. Additionally, the components of a distributed application are less likely to undergo modifications. Leveraging these stable conditions, it is advisable to activate caching mechanisms specifically designed to enhance performance.

This guide provides a comprehensive checklist of configurations recommended for review and adjustment in a production setting. Implementing these configurations can significantly optimize performance and enhance the user experience.

Base Configuration¶

The Base Configuration encompasses all parameters that the Module Suite utilizes for its core functionality, as well as for all installed extension packages. To access the basic configuration page, navigate to the Content Server's administrative interface and select the Base Configuration link within the AnswerModules administration settings.



Configuration Parameters¶

The parameters available for configuration fall into two primary categories:

1. **Performance Optimization Parameters:** These parameters are specifically designed to enhance system performance in production environments. By adjusting these settings, you can ensure efficient operation and optimized resource use.

2. **Usage-Based Tuning Parameters:** These settings allow for the fine-tuning of the Module Suite to align with the actual usage patterns of the tool. Given the diverse range of use cases for the Module Suite, it is understandable that not all parameters will be relevant to every implementation.

Below is a list of the parameters, beginning with those that have the most significant impact on performance. For each parameter, we provide the recommended baseline value or possible alternatives, accompanied by a brief explanation of their purpose.

Performance Optimization Parameters Table¶

The following table outlines key configuration parameters, their recommended baseline values or alternatives, and a brief description of each parameter's purpose.

Parameter Name	Recommended Value	Alternatives	Description
`amcs.amsui.volumeCache`	`true`	`false`	Enables the caching of the portion of the Content Script Volume related to

Parameter Name	Recommended Value	Alternatives	Description
			enhancements to be applied to the Smart View.
`amcs.amsui.volumeCache.ttl`	`3600`	NA	The duration of the entries in the cache above in seconds (must not exceed 30 days).
`amcs.cache.connectionString.default`	A space separated list of hostname and port pairs, e.g., `myserver:8512 myserver:8513`.	NA	Module Suite uses memcache for implementing caching at various levels; it is essential this configuration contains accurate values.
`Enable/Disable Module Suite internal cache`	`unchecked`	NA	Allows Module Suite to cache the Content Script Volume and other objects.
`Enables the Beautiful Webforms View Template Cache`	`checked`	NA	Caches the information related to the skin associated to each form view.
`Store Static Variables in memory`	`checked`	NA	Caches the information related to the Script static variables.
`amcs.core.callbacksUserIDs`	`1000`	`empty`	A comma-separated list of user IDs for whom it is possible not to track sync events.

Please ensure these values are accurately reflected in your Module Suite configuration to optimize performance and functionality.

Usage-Based Tuning Parameters Table¶

The following table provides detailed information on key usage-based tuning parameters, including recommended settings, alternatives, and descriptions to guide adjustments based on specific usage scenarios.

Parameter Name	Recommended Value	Alternatives	Description
`Enable/Disable Asynch events management`	`checked`	NA	This checkbox disables the feature that tracks events on the Content Server and populates the queue for the Asynchronous Job handler to process them later. Inspect the content of the CSEvents folder in the Content Script Volume to determine usage. If no Content Scripts are found, the feature is not used and can be safely disabled.
`List Nodes API for complex and convoluted ACLs`	`unchecked`	`checked`	This setting should be considered if working in an environment with many nested groups and experiencing problems listing content in spaces or folders due to complex and convoluted ACLs.
`xECM for Everything`	`unchecked`	NA	Enable this feature only if you have deployed a Module Suite SPI adapter leveraging the xECM for Everything functionality.
`amcs.core.callbackSynchEventsEnabled`	`false`	`true`	This property enables the feature that tracks events on Content Server and triggers the execution of configured Content Scripts. Inspect the content of the CSSynchEvents folder in the Content Script Volume to determine usage. If no Content Scripts are found, the feature is not used and can be safely disabled.

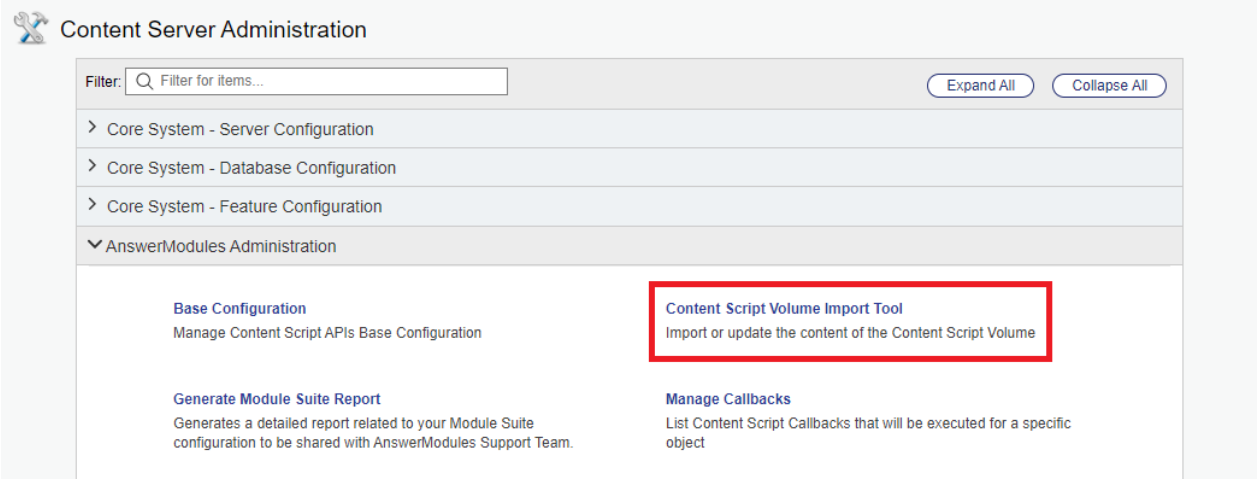
Please adjust these settings based on the actual usage patterns and requirements of your Module Suite implementation.

Content Script Volume¶

The Content Script Volume plays a crucial role in the performance of productive environments. Specifically, before version 3.6 of our software, failing to import the portion of the volume dedicated to enhancements for the SmartView could lead to unnecessary database queries by the Module Suite. This inefficiency can be effectively addressed by ensuring the SmartView-related content of the Content Script Volume is fully imported.

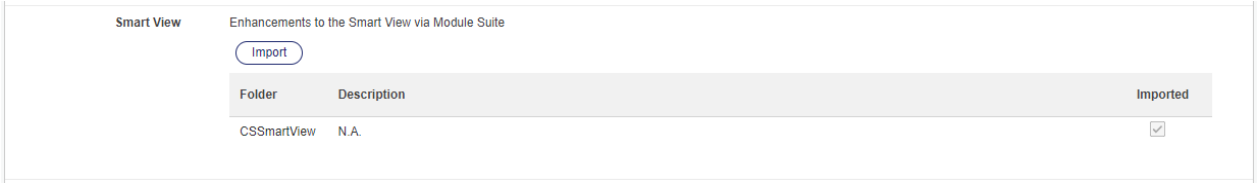
Importing SmartView Enhancements¶

To import the necessary enhancements for SmartView, utilize the Content Script Import Tool.



Through this tool, you can achieve a complete setup by following these steps:

1. Navigate to the "SmartView" section within the Content Script Import Tool.
2. Click the import button to begin the process.



For detailed instructions and more information, visit the Content Script Import Tool documentation.

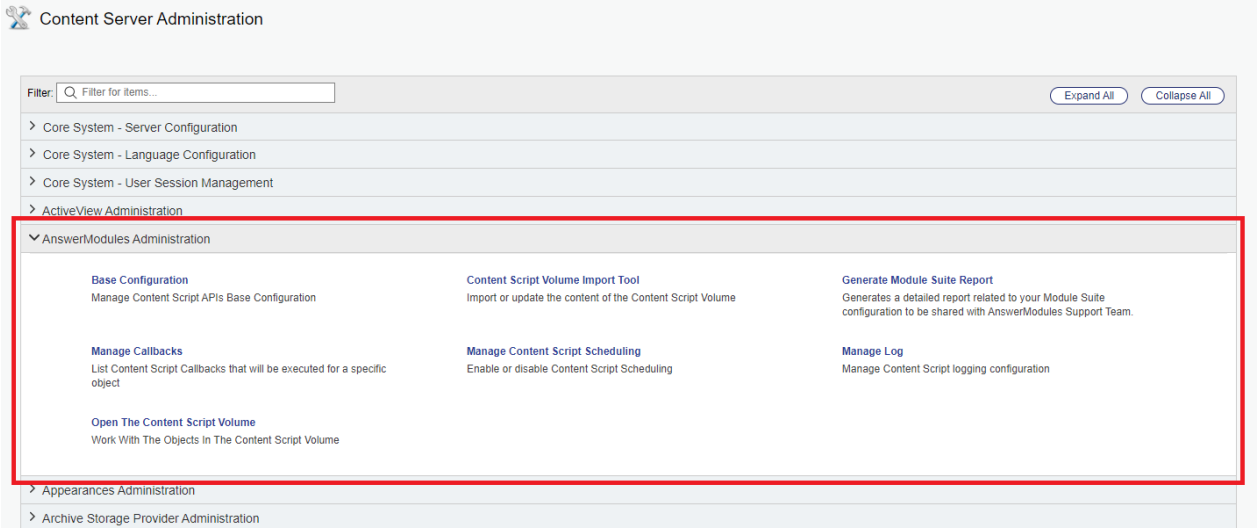
By following these steps, you can enhance your system's performance by eliminating redundant database queries related to SmartView.

Administration

administration

Module Suite Administration Tools¶

Settings and administration tools specific to ModuleSuite components can be accessed from the Content Server Administration pages.



Detailed information related to the single tools and configuration pages is provided in the following sections.

Base Configuration¶

The Base Configuration page provides access to:

- Software activation status
- Content Script Volume Library version
- ModuleSuite database maintenance utilities
- global configuration of the Content Script engine, and configuration of the single API services
- configuration of custom Content Script extension modules

Configuration Export and Import

Since Module Suite version 3.2, the Base Configuration settings can be export and/or imported using standard Content Server administration tools.

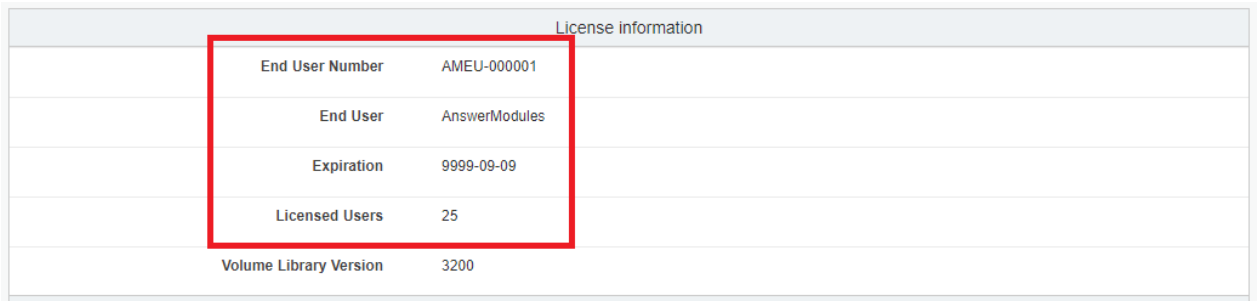
Base Configuration updates and system restarts

Since Module Suite version 3.2, most changes to the Base Configuration settings **no longer require a restart**.

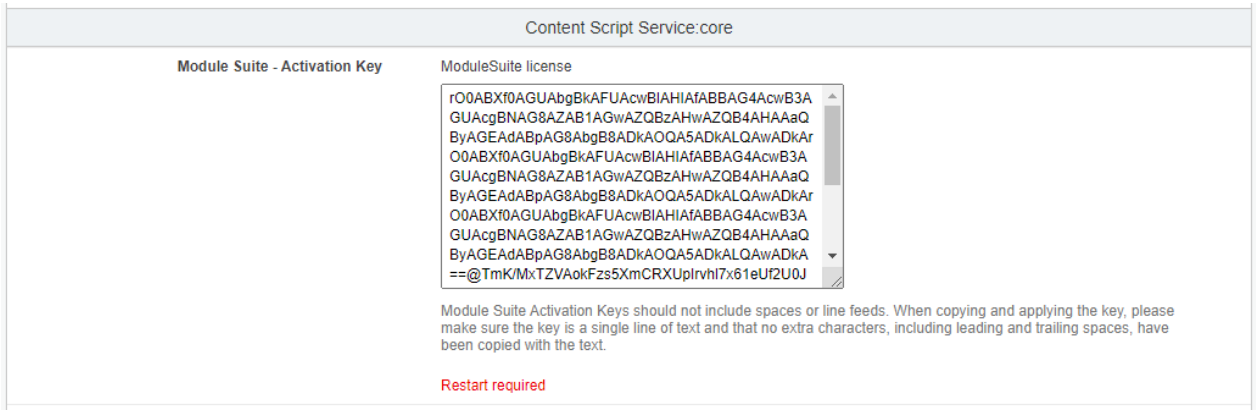
Nevertheless, certain specific features will still request a system restart: they are flagged in the Base Configuration pages with a "restart required" label.

Software activation key status¶

The activation status of the Module Suite software can be found in the first section of the Base Configuration page.



The actual activation key can be found in the "Core" section of the configuration page.



Apply or update the activation key

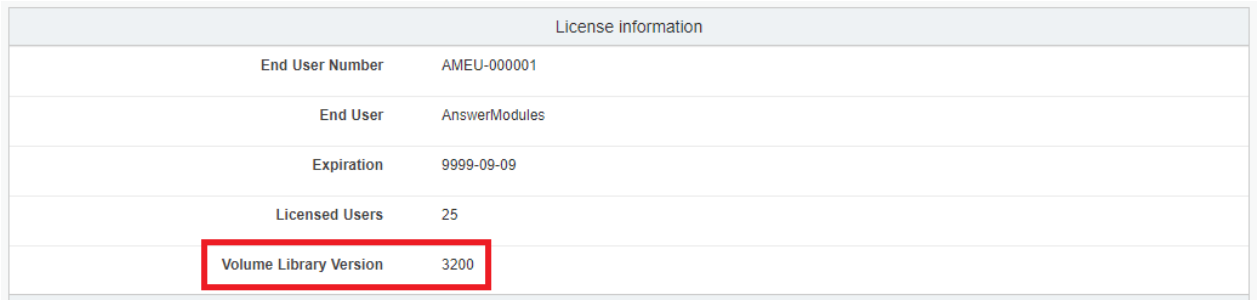
The activation key can be manually applied as described in the "Activate Module Suite by manually setting the activation key" section in the installation.

Alternatively, since Module Suite version 3.2, the Base Configuration settings can be exported and/or imported using standard Content Server administration tools. This includes the Module Suite activation key.

See the "Activate Module Suite by importing the activation key" section in the installation guide for further details.

Content Script Volume Library¶

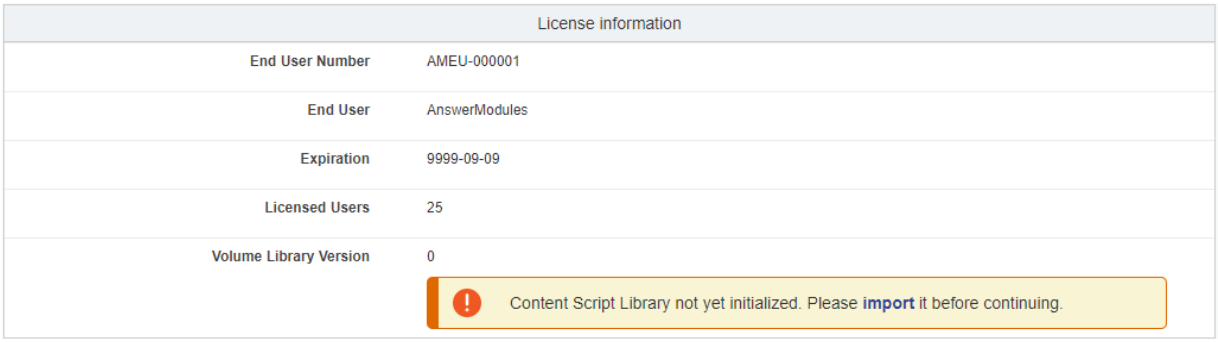
An indication of the current Content Script Volume library version is shown Within the top section of the Base Configuration page.



Library not yet initialized warning

Upon initial installation, the Volume Library will appear as "not yet imported" and a warning message will be shown.

To finalize the installation, import the Volume Library through the Content Script Volume Import Tool.



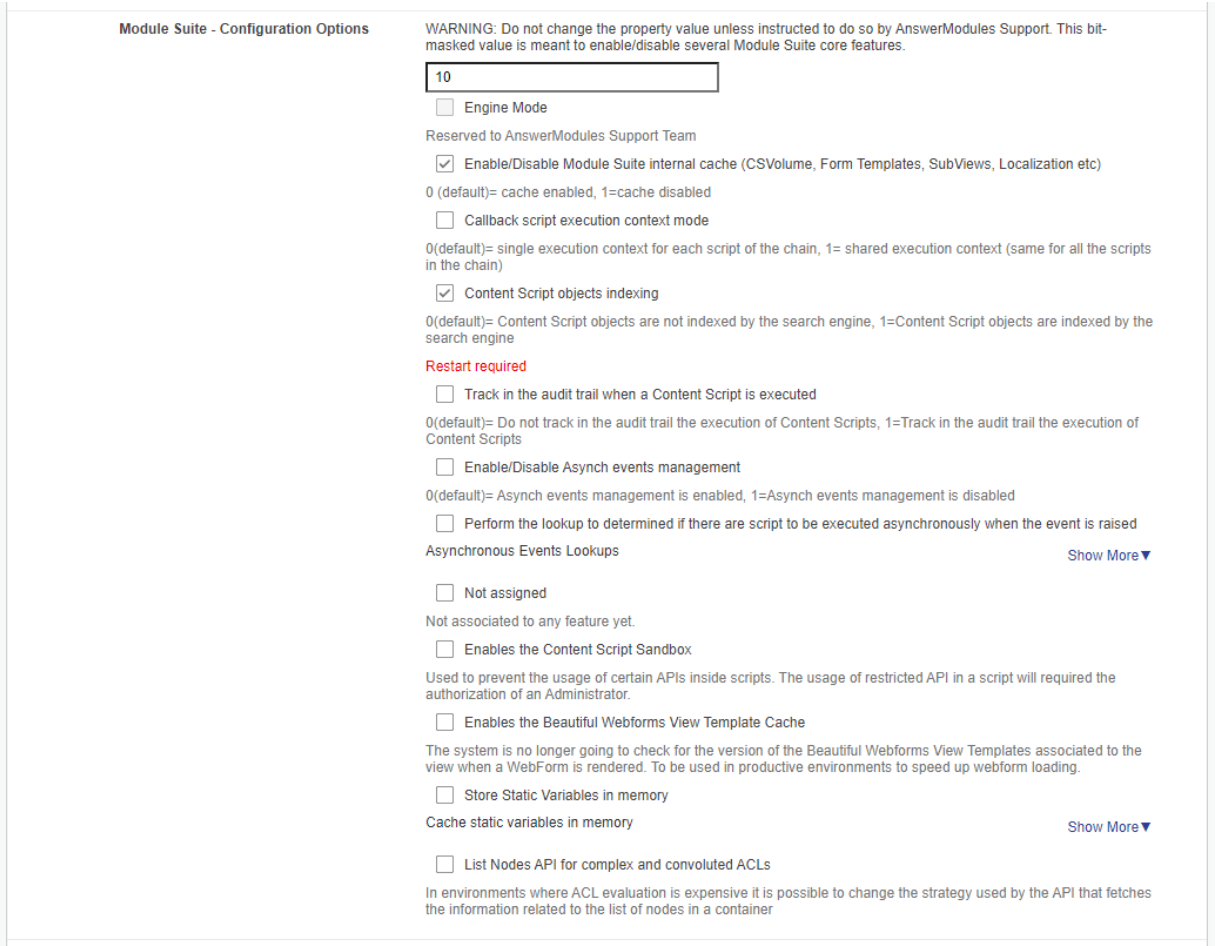
Enable / Disable Module Suite features¶

The amcs.core.debugEnabled is a "core" configuration bitmask you can use to customize your Module Suite instance enabling/disabling certain core features at once. Each bit in the mask represent a different feature that can be enabled (0) or disabled (1), or switched between different execution modes.

Enhanced management in version 3.2

Since Module Suite version 3.2, each separate feature within the "core" configuration bitmask can be controlled through checkbox selectors.

Upon toggling the single options, the overall bitmask decimal value will automatically be updated.



Additionally, the system will request a restart only in case where a feature that requires it is updated.

Here below a reference for the meaning of each bit in the mask.

Position	Meaning	Valid values	Decimal value
1	RESERVED	0	
2	Enable/Disable Module Suite internal cache (CSVolume, Form Templates, SubViews, Localization etc)	0 (default)= cache enabled, 1=cache disabled	2
3	Callback script execution context mode	0(default)= single execution context for each script of the chain, 1= shared execution context (same for all the scripts in the chain)	4
4	Content Script objects indexing	0(default)= Content Script objects are not indexed by the search engine, 1=Content Script objects are indexed by the search engine	8

Position	Meaning	Valid values	Decimal value
5	Track in the audit trail when a Content Script is executed	0(default)= Do not track in the audit trail the execution of Content Scripts, 1=Track in the audit trail the execution of Content Scripts	16
6	Enable/Disable Asynch events management	0(default)= Asynch events management is enabled, 1=Asynch events management is disabled	32
7	Perform the lookup to determined if there are scripts to be executed asynchronously when the event is raised	0(default)= Any "interesting" event for Asynch events management is tracked in the Distributed Agent queue and the lookup required to determine if there are scripts to be executed is performed later on by the same DA worker that manages script execution, 1=The lookup required to determine if there are scripts to be executed asynchronously given the registered event is executed when the event is raised. The information is passed to the DA queue only if the lookup finds that there are scripts that need to be executed	64
8	RESERVED	0	128
9	Enables the Content Script Sandbox (disabled by default)		256
10	Enables the View Template Cache (The system is no longer going to check for the version of the Beautiful Webforms View Templates associated to the view when a WebForm is rendered)		512
11	For every Content Script, it is possible to define a set of static, precompiled variables whose values will be available when the script is executed. The framework supports the definition	0(default)=cache disabled. 1=cache enabled	1024

Position	Meaning	Valid values	Decimal value
	of these variables by means of a second script, whose outcome is the data map containing the values. For performance reasons, this second script is executed only when it changes (or when execution is explicitly forced by an editor), and the results are stored as part of the script object. The information is retrieved from the Database upon execution when you execute a subscript using the runCS API or you retrive this information using the getCSVars API on a CSScript object. In situation in which the Database is under stress or the retrieval of this information does not perform as expected it is possible to configure the framework so that this information is cached in memory.		
12	In environments where ACL evaluation is quite expensive it is possible to change strategy used by the API that fetches the information related to the list of nodes in a container	0(default)=standard strategy (should work well in most of the cases), 1=Alternative strategy (due to complex and convoluted ACLs)	0
13	When the configuration is exported using the standard export feature of the Content Server, all the actual values of the secret configuration parameters specified on this page are omitted. You can let the system export them by checking this configuration option.	4096	
14	Allow the Script Manager to inject support paths information into the script execution context. It may be required by some extension packages to work properly (e.g., rend)	0	8192
15	Enable "xECM 4 Everything" to seamlessly create SPI adapters via our Module Suite. This promotes efficient data exchanges and streamlines	0	16384

Position	Meaning	Valid values	Decimal value
	workflows across varied platforms. Enhance flexibility and interoperability for a unified digital workspace.		
16	In environments where a very large number of objects are managed, this option ensures a proper conversion between Oscript's 64-bit integer representation and Java's long type.	0	32768
17	Seal Content Script Versions: When activated, this setting blocks the creation of new versions of Content Scripts by users with standard permissions. Only administrators and those explicitly allowed to create scripts can modify versions, helping to ensure the integrity of your productive environments.	0	65536
18	Limit Administrators: This feature is recommended on productive systems. When activated, it restricts System Administration users from creating or updating Content Scripts unless they are members of the Privilege group.	0	131072

Example of valid configuration values:

- Enable Content Script indexing while disabling Module Suite cache: 8+2 = 10
- Enable Content Script execution audit trail while disabling Asynch events management: 16 + 32 = 48

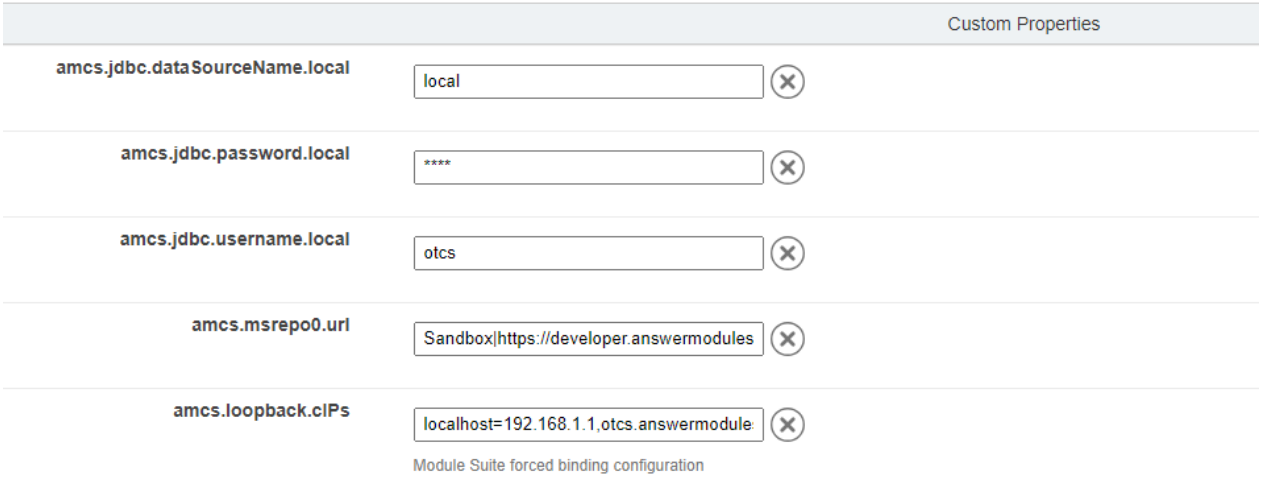
Select default IP address¶

It is quite common for Content Server services to be installed on servers that have multiple network interfaces associated with different IP addresses.

Sometimes it is desirable to control which interface or IP address Module Suite uses for external communication (for example with the Content Script extension's **csws** service). In such a situation you can use an additional custom configuration parameter in the base configuration to control interface binding. In fact, the Custom property ***amcs.loopback.cIPs*** allows you to bind an IP address to its server host address. Multiple mappings are supported.

IP Mapping Configuration

`hostname.domain.com=192.168.100.100`.



Keep in mind that IP addresses must be valid and assigned to one of the server's network interfaces.

SASL Memcache Authentication Support¶

Module Suite 3.7.0 introduces support for SASL memcache authentication. When enabling this feature on OTCS, follow these important steps:

Single Thread Client Configuration

Ensure that the cache is configured to use a single thread client. To do this:

1. Navigate to the Module Suite base configuration.
2. Locate the `amcs.cache.mode.default` property.
3. Set its value to `single`.

Configuration Reload Required

After enabling SASL authentication on OTCS, you must save the Base Configuration to force a configuration reload.

Steps to Enable SASL Memcache Authentication¶

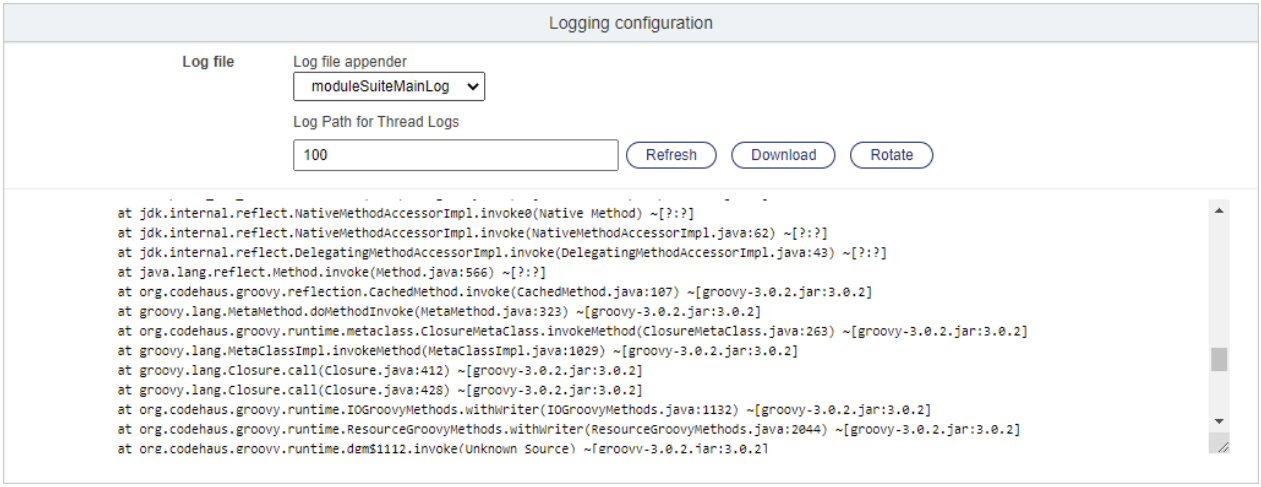
1. Configure the cache to use a single thread client as described above.
2. Enable SASL authentication in your OTCS settings.
3. Save the base configuration to apply the changes.

Logging administration¶

The Content Script logging utility allows administrators to:

- access the log file without the need to log on to the server where the log resides
- configure the log level of Content Script objects that include logging instructions.

Accessing the log file¶



When opening the utility, the last lines in the log file will be automatically shown to the user. It is possible to perform the following actions:

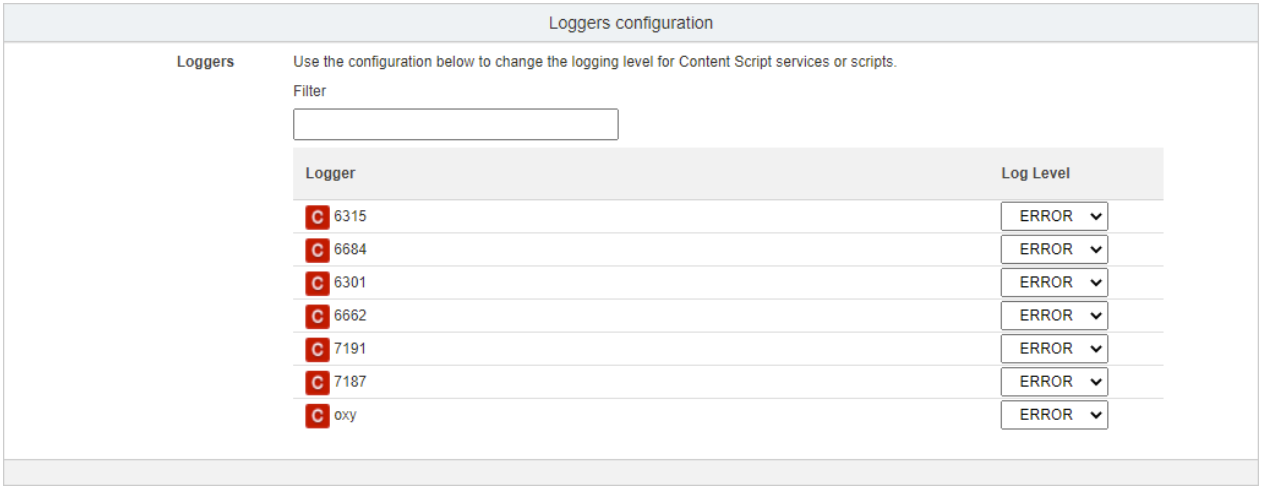
- **Refresh** the screen to check for changes in the log
- **Rotate** the log (replaces the log file with an empty one)
- **Download** the complete log file

Log level configuration¶

The log level management section allows to change the logging level for each single Content Script object, at runtime.

Log level selection is progressive: setting the log level to a certain threshold will instruct the system to log all entries of that specific level, in addition to any entry of higher severity. For example:

- when setting the log to DEBUG, INFO and ERROR entries will also be logged
- when setting the log to ERROR, INFO and DEBUG entries will **not** be logged.



No restart required

Logging level can be changed at runtime without restarting the Content Server.

Past logs below threshold cannot be recovered

Note that changing the log level will only affect any future logging operations.

Past log entries below the original threshold are discarded and cannot be recovered.

Default log level is restored on system restart

Changes to the log level performed through this tool do not survive a restart of the OTCS services.

Upon restart, the log level will be set back to the default value (typically "ERROR").

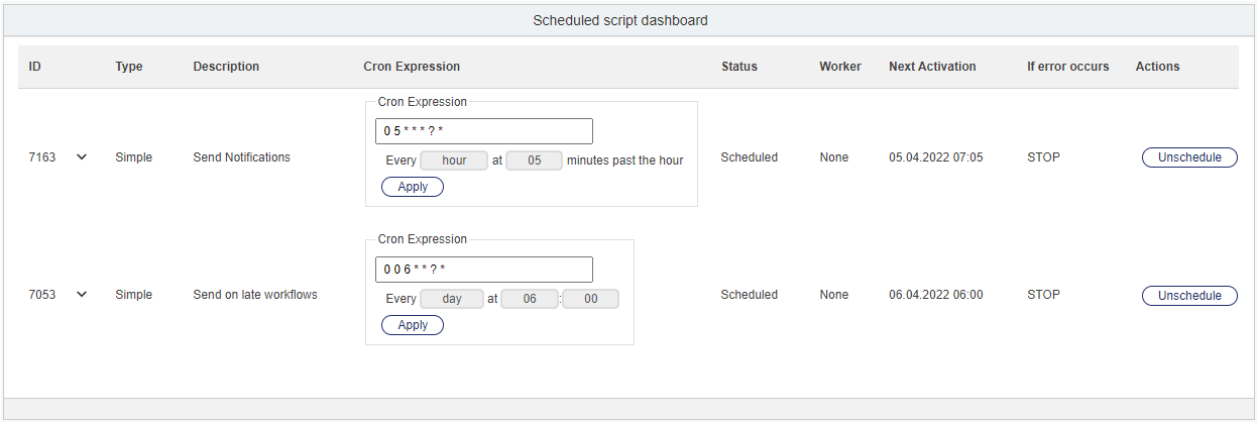
Where is my log ?

The log management utility is **not centralized**: when running on a clustered environment, it is important to note that the utility will only show log contents and loggers configuration **for the current server** that is being accessed.

Logging, on the other hand, is specific to the single server/instance where the operation triggering the log is performed. It is important to keep this in mind when analyzing the log data, as an operation could have been executed on different servers. For example, logging entries related to scheduled scripts or asynchronous callbacks will typically be found on the servers where the Distributed Agents are set to run.

Scheduling management utility (Manage Scheduling)¶

The Content Script Scheduling administration panel provides a quick overview of the Content Script objects that are queued for scheduled execution, together with the next fire time, the expression used to calculate the execution schedule, and generic information related to the object itself. The object menu allows to easily access the node standard functions.



An **unschedule** utility allows to stop the scheduling of the corresponding script.

Configuration

The complete set of configuration options for Content Script scheduling (as well as impersonation settings) are available through the Content Script editor Administration tab

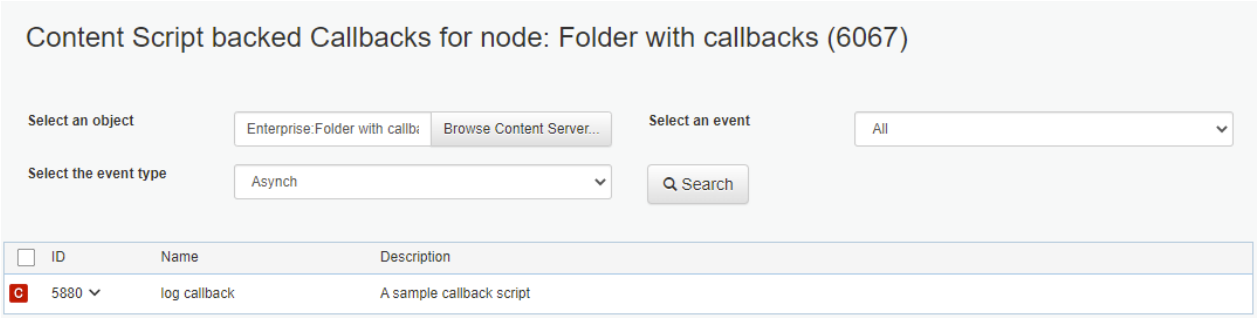
Callbacks management utility (Manage Callbacks)¶

The Callbacks management utility provides a tool to verify in every moment what Content Script callbacks will be executed for specific objects in response to specific event types.

The utility provides a set of filters that allow to identify:

- the target object
- the specific callback type(s) to be analyzed
- the callback mode (synchronous or asynchronous)

Based on the filter, the result will show a list of affected nodes.



Module Suite Report utility¶

The Module Suite Report utility allows the administrators to generate a report containing information relevant to the installation, configuration and execution of Module Suite.

This is especially useful in case of issues to share details regarding the environment with product support representatives.

The generated report is in text format, as show below:

```
Module Suite Report
    Generated: 04/04/2022 14:05:40

Usage Stats

    Active users: 1
    Licensed users: 25

Database

    Server DBMS: POSTGRESQL

Content Server Modules

    module/
        ansbwebform_3_2_0
        anscontentscript_3_2_0
        anscontentsmartui_3_2_0

Base Configuration

    BaseConf

        adlib
            amcs.adlib.activeProfiles = default
            amcs.adlib.inputFolder.default =
            amcs.adlib.jobInputFolder.default =
            amcs.adlib.jobOutputFolder.default =
            amcs.adlib.resultTimeout.default =
            amcs.adlib.resultPollInterval.default =
            amcs.adlib.xmlHeaderLine1.default = <?xml version="1.0" encoding="ISO-8859-1" ?>
            amcs.adlib.xmlHeaderLine2.default = <?AdlibExpress applanguage="USA" appversion="4.1.0"
```

The Content Script Volume¶

The **Content Script Volume** is a Content Server volume automatically created upon module installation.

The volume is used to store objects for various purposes. Among others, in the Content Script volume we may find:

- **System Objects:** Objects necessary for the correct execution of different Module Suite components. These objects should not require modification in normal cases.
- **Configuration Objects**: Objects used to configure specific functionalities
- standard UI customization
- event callback configuration
- custom column data sources

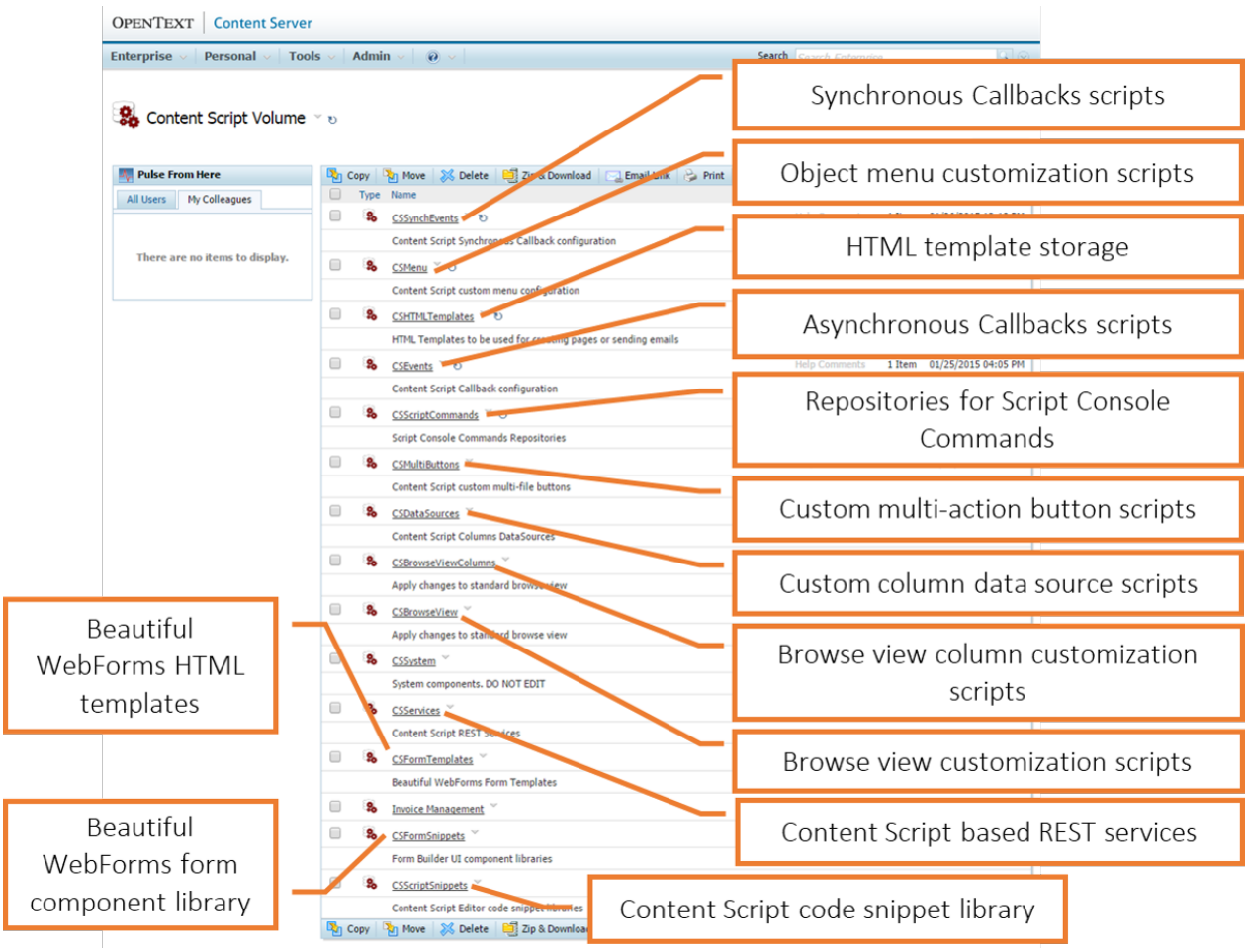
- **Template Objects**: Various sorts of objects to be used as templates, such as:
- Content Script code snippets
- Beautiful WebForms form templates
- Beautiful WebForms form components
- HTML view templates
- ...
- **Service Scripts**: Scripts executed as service endpoints
- Content Script backing REST services
- ...

Whenever possible, a **convention-over-configuration** approach is adopted in the Content Script Volume: simply placing a specific object in a specific position will be enough to alter in some way the behavior of some functionalities.

For this reason, a set of **predefined containers** is available in the volume, each one meant for a specific purpose. Here after is a view of the Content Server Volume.

The following sections will explain the purpose of each of the Containers.

> **How should I organize my volume ?**
>
> Even though the Content Script Volume has a predefined container structure, it is not unusual to have custom user data to be stored in the volume. Users are encouraged to use the volume to store custom templates and configurations, for example.



CSSystem¶

The **CSSystem** container is dedicated to Module Suite system components. The contents in this location should not require editing except for very specific reasons.

CSFormTemplates¶

This container is dedicated to HTML templates associated to **Beautiful WebForms Views**.

It will be covered in detail in the sections dedicated to Beautiful WebForms *(/working/bwebforms/sdk/#csformtemplates)*.

CSHTMLTemplates¶

The **CSHTMLTemplates** is a container dedicated to general-purpose HTML templates that could be necessary throughout Content Script applications.

As previously seen, Content Script can be used to create various types of output, including web pages and document. Additionally, a few services (such as the **mail** service) can use templates to perform their job.

It is usually discouraged to place HTML templating code directly within Scripts: the suggested approach is to separate the presentation templates from the underlying business logic, and to store it somewhere else on Content Server, where it can be reused across applications.

The **CSHTMLTemplates** container is available for developers as a common storage for templates necessary in their applications.

CSFormSnippets¶

The **CSFormSnippets** container is dedicated to the libraries of components that are available to build Beautiful WebForms views.

It will be covered in detail in the sections dedicated to Beautiful WebForms.

CSScriptSnippets¶

The **CSScriptSnippets** container features a two-level structure identical to the one described for the CSFormSnippets container, except that the objects stored here are not form components but Code Snippets to be used to simplify the creation of new scripts in the Content Script Editor.

As for the Form Snippets, new families and components added in this container will automatically be available in the Code Snippet library of the Content Script Editor.

administration installation upgrade

Content Script Volume Import Tool¶

> **Major change in version 3.2.0**
>
> Since Module Suite version 3.2.0, there have been major changes in the way the content of the Content Script Volume is managed.

Overview¶

Module Suite's components behaviour and functionalities can be modified and extended by manipulating the content of the Content Script Volume (a Content Server's Volume created when installing the Content Script module).

Prior to Module Suite version 3.2, all Content Script Volume resources had to be necessarily imported in the Volume, with no exceptions. Starting with version 3.2, Module Suite is capable of using certain resources (CSFormSnippets, CSScriptSnippets, CSPageSnippets) directly from

the Module installation folders on the filesystem, without the strict need to "materialize" them in the Content Script Volume. This approach allows to avoid the overhead of importing certain resources if the administrator does not plan to customize them, but it optionally allows to "materialize" them in the Volume if needed.

This new approach allows to significantly reduce the effort required in validating the content of the Content Script Volume and solving conflicts in case of updates, since if the resources have not been materialized, the update will be transparent for the users (the library in the new Module version will replace the old one).

As a result of this new approach, the CSVolume administration tools have been reorganized and updated.

All **"system critical"** resources are now automatically imported (and updated) though a **Volume Library** management utility. This currently includes:

- CSSystem
- CSServices
- CSi18n
- CSImports
- CSPageTemplates
- CSFormTemplates
- CSPageSnippets **(folder structure only)**
- CSFormSnippets **(folder structure only)**

Optional **"feature activation"** resources can be boostrapped on-demand through a dedicated set of utilities ("Module Suite Features"). This will include reources such as:

- CSEvents
- CSSynchEvents
- CSMenu
- ...

The following resources are **not** imported by default, as the system is capable of using them from the original library on the filesystem :

- CSFormSnippets
- CSPageSnippets
- CSScriptSnippets

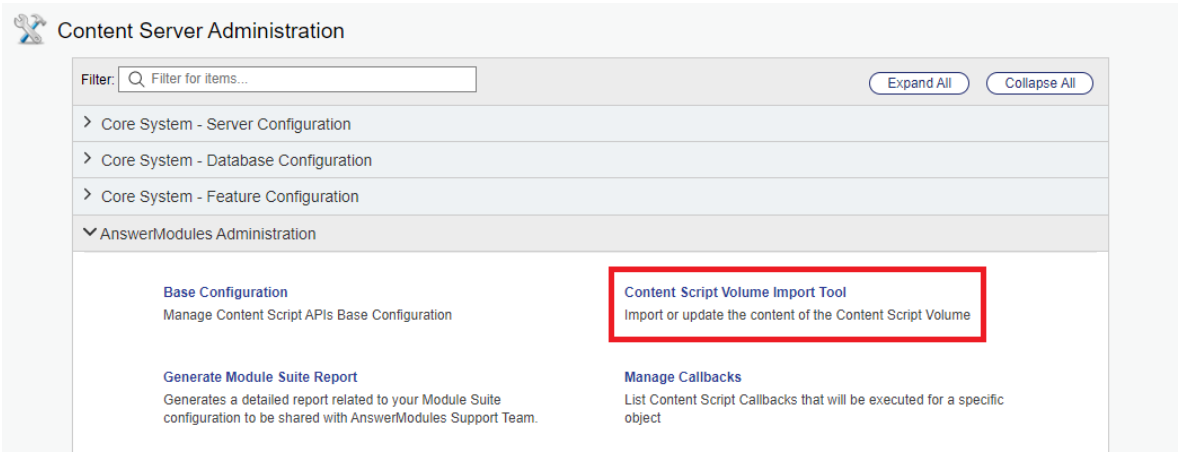
Nevertheless, it will be possible to "materialize" them locally using the Volume Conflict Resolution utility.

This section will describe the available tools designed to simplify the management of the content of the Content Script Volume, handling new imports, updates and conflicts resolution.

Accessing the Content Script Volume Import Tool¶

In order to access the Content Script Volume Import Tool:

- ☑ As the system Admin user, open the Content Server Administration pages.
- ☑ Locate the **AnswerModules Administration** section. Within this section, open the **Content Script Volume Import** tool.

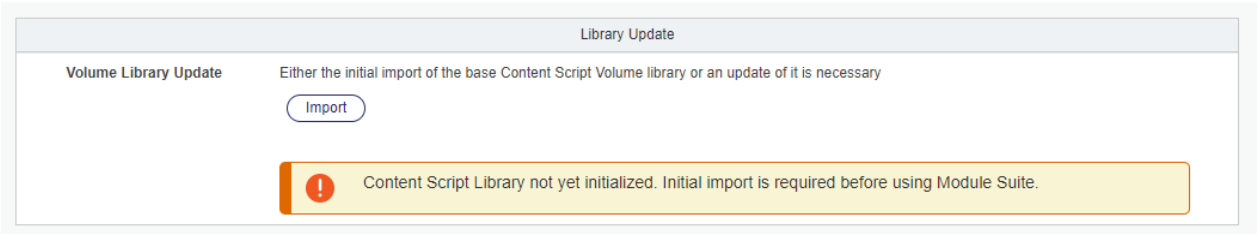


Volume Library utility¶

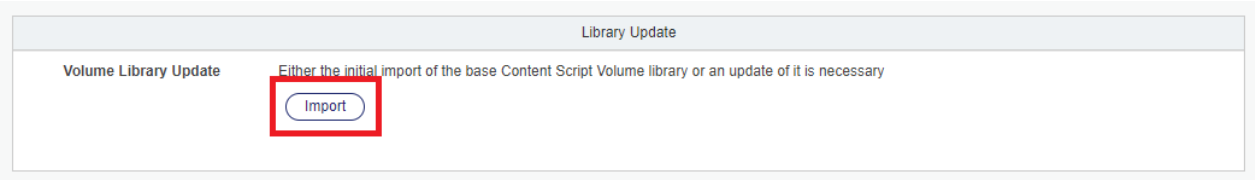
One of the main features of the Content Script Volume Import Tool is to assist the OTCS system administrator in the management of the core Volume Library. This library includes all Content Script Volume elements that are critical for the execution of Module Suite. As part of the initial installation process, the Volume Library should always be imported in the Content Script Volume.

When opening the Content Script Volume Import Tool, the "Volume Library" section will occasionally show up as the very first section in the page.

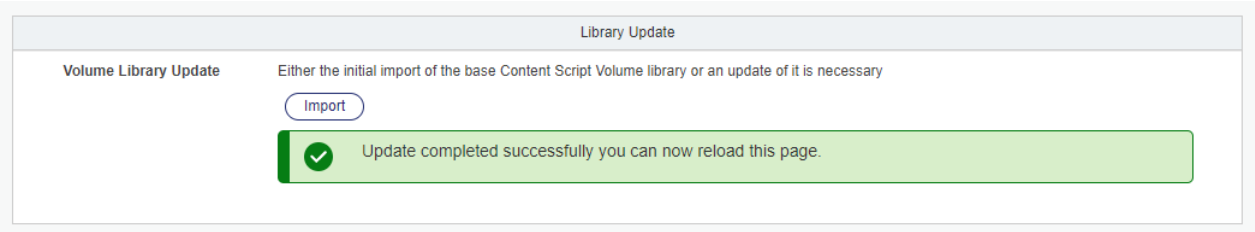
On initial installation, an alert message will notify the administrator that the Volume Library has not been imported yet.



In case of subsequent upgrades, the Content Script Volume Import Tool should always be checked to verify the presence of changes to be imported. In case of pending updates, the Library Update section will show, together with an "import" button.



Once complete, the Volume Library import will prompt the user to refresh the page.



If the "Volume Library" section not shown, the Volume Library is up-to-date and no action should be taken.

Module Suite Features utilities¶

The Content Script Volume Import Tool is also used to control the root Content Script Volume elements required to activate certain functionalities of the Module Suite.

For each feature, a dedicated section of the tool will allow to automatically bootstrap the related Content Script Volume content. A table with the following values is shown:

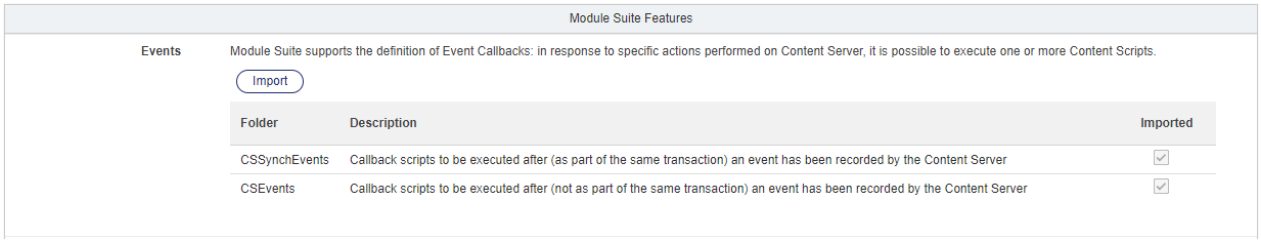
- **Folder** : The corresponding CSVolume root folder required to activate this feature
- **Description** : Additional usage details of the folder
- **Imported (y/n)** : A status flag showing wether the resource has been already imported in the system

Additionally, each section will include an **import** button.
When using the import, the specific folders (and the initial content, if necessary) will be automatically set up on the system. If the import function is used for resources that were already imported, any missing resource will be initialized, but existing resources will not be touched.

At the present moment, the following features can be activated:

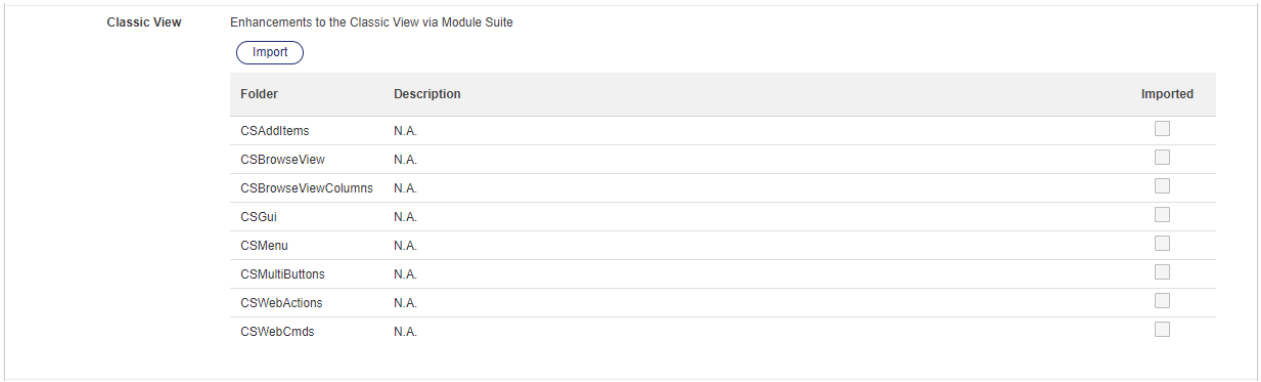
Events¶

Manages all resources necessary to use the Content Script Callbacks (synchronous and asynchronous).



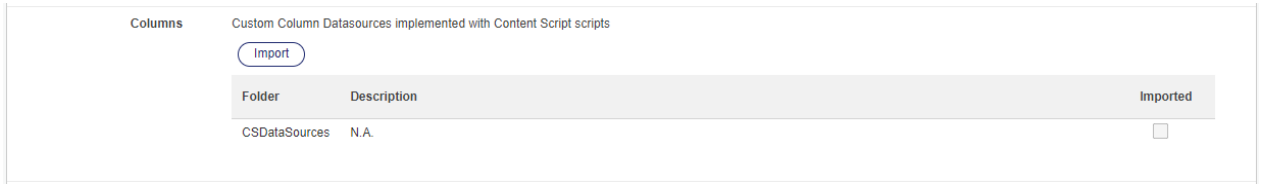
Classic View¶

Manages all resources necessary to use the Content Script extension for Classic UI features.



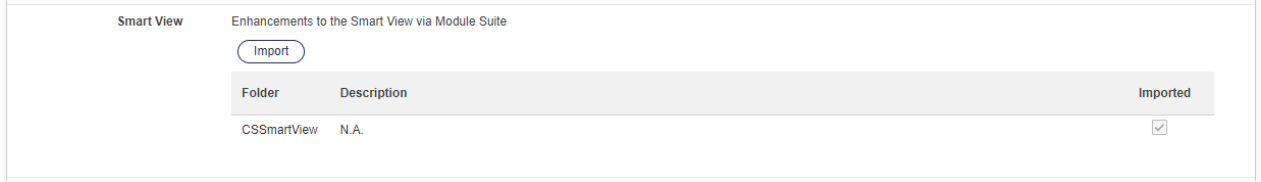
Columns¶

Manages the resources necessary to create Content Script column datasources.



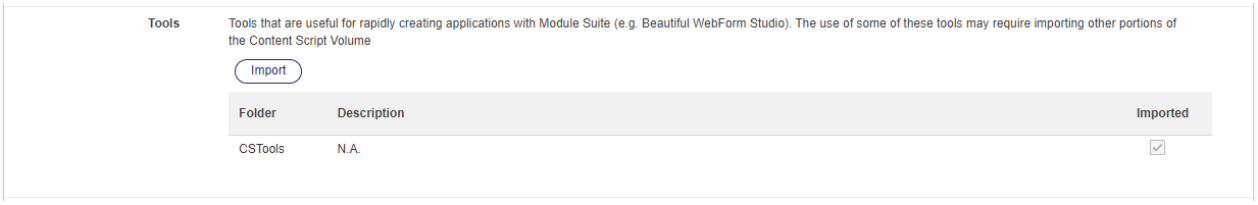
Smart View¶

Manages all resources necessary to set up Smart UI overrides using the Smart Pages capabilities.



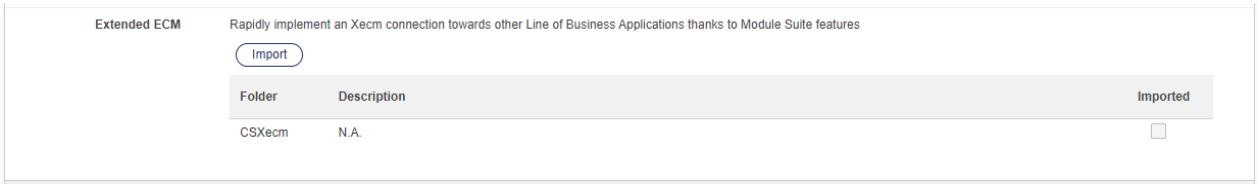
Tools¶

Imports the available Content Script Tools (e.g. BWF Studio, PDF Viewer, ...)



Extended ECM¶

Sets up the resources required to use the "xECM for Everything" capabilties.



Volume's Conflicts Resolution utility¶

Since Content Script Volume objects are accessible and customizable by the Module Suite administrators and developers, it is possible to generate conflicts between the customized versions and the new/updated versions included in Module Suite upgrade packages.

The Content Script Volume Import Tool includes a utility that lets the administrators:

- identify and resolve any version conflicts in the Content Script Volume.
- materialize certain resources in the Content Script Volume in order to allow for local customization.

Upon opening the page, the tool will automatically trigger the analysis of all the objects present within the Content Script Volume. This operation might require some time to complete, depending on the content of the Content Script Volume.

Once the analysis is complete, the utility will present with a tree view of all relevant Content Script Volume resources.

The following resources will **always** be present, regardless of their status:

- CSFormSnippets
- CSScriptSnippets
- CSPageSnippets

All other containers will only be present in case of conflicts with the corresponding filesystem resources.

Identifying conflicts¶

For each resource, depending on the status, the administrator will have a choice to handle the conflict.

- For CSFormSnippets, CSScriptSnippets, and CSPageSnippets that **have not** been materialized on the system, the utility will show a "Not overridden" status. Importing the object will result in the object being created in the system. This will override the original library object when that specific resource is used.
- For CSFormSnippets, CSScriptSnippets, and CSPageSnippets that **have** been materialized on the system, the utility will show any conflicting situation (for example, if the version on the system is different from the one in the original library). It is up to the administrators to solve or ignore these conflicts.
- For all other resources (which the system is not capable of using if not imported in the Content Script Volume), that **have not** been imported, the utility will show a notice that the object is available to be imported.
- For all other resources (which the system is not capable of using if not imported in the Content Script Volume), that **have** been imported, the utility will show any conflict status (newer version available for import, conflict, etc..). It is up to the administrators to solve or ignore these conflicts.

Import options¶

The utility presents two distinct options to import the selected objects.

- **Import** option: this will result in the materialization of the selected resource(s) in the Content Script Volume. In case the resource was already present in the Volume, it is skipped and the local changes are not reverted.
- **Override and update** : this will result in the materialization of the selected resources(s), regardless of the presence of a local version in the Content Script Volume. This operation will override any changes performed locally.

Content Script

Getting Started with Content Script¶

This guide provides a quick introduction to **Content Script** and helps you get started with creating custom scripts and extending Content Server functionality.

What is Content Script?¶

Content Script is a **Domain-Specific Programming Language (DSL)** for OpenText Content Server. It enables you to programmatically interact with Content Server and external systems, automate business processes, create custom web interfaces, and extend Content Server functionality without traditional OScript development.

For a comprehensive overview, see the Content Script Architecture.

Key Components¶

The Content Script module includes:

- **Content Script Objects** - Document-class objects that contain executable Groovy-based scripts
- **Web-based IDE** - Integrated development environment for creating and editing scripts
- **API Services** - Comprehensive set of services for interacting with Content Server and external systems
- **Event Callbacks** - Synchronous and asynchronous event handlers for Content Server events
- **REST API Support** - Expose scripts as RESTful web services
- **Workflow Integration** - Use scripts as workflow steps with routing capabilities
- **SDK** - Toolkit for creating custom Content Script services

Quick Start Guide¶

1. Understanding the Basics¶

Start by reading the Content Script Architecture to understand:

- How Content Script works

- API Services and execution context
- Script execution modes and outputs

2. Creating Your First Script¶

Learn how to create and manage Content Script objects:

- Content Script Objects - Creation, properties, static variables, scheduling, and impersonation
- Content Script Editor - Learn how to use the web-based IDE to write and test scripts

3. Learning the Language¶

Understand the Content Script language syntax:

- Content Script Language - Groovy-based syntax, operators, closures, and programming constructs

4. Working with APIs¶

Explore the available API services:

- Content Script Architecture - Complete overview of all API services (docman, workflow, search, users, mail, etc.)

5. Event-Driven Programming¶

Implement event callbacks:

- Events and Callbacks - Synchronous and asynchronous event handlers for Content Server events

6. Extending Functionality¶

Explore advanced integrations:

- REST API - Expose scripts as RESTful web services
- Workflow Integration - Use scripts as workflow steps
- Rendition - Programmatically invoke external rendition engines
- SAP Integration - Integrate with SAP ERP systems
- SDK - Create custom Content Script services

Content Management Object

Content Script objects are document-class objects on Content Server.

Content Scripts are **restricted** objects: as such, users must be **enabled** to the creation of new objects through the Administration pages.

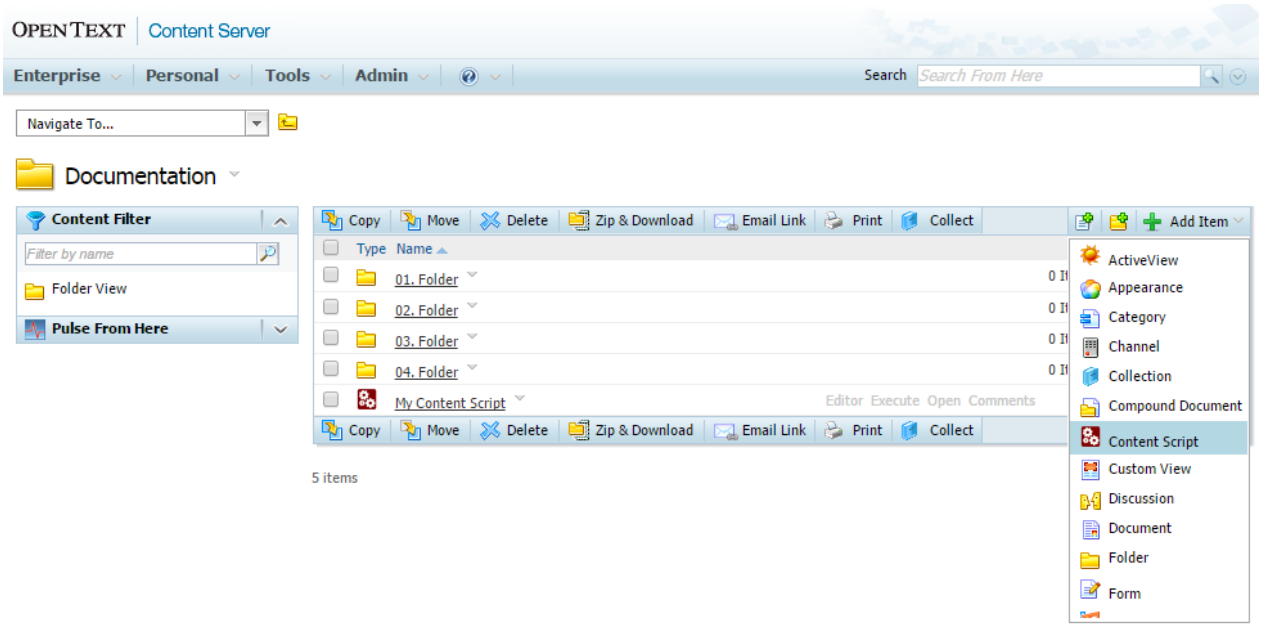
Content Scripts are executable objects, and the **execution** is the default action associated to the object.

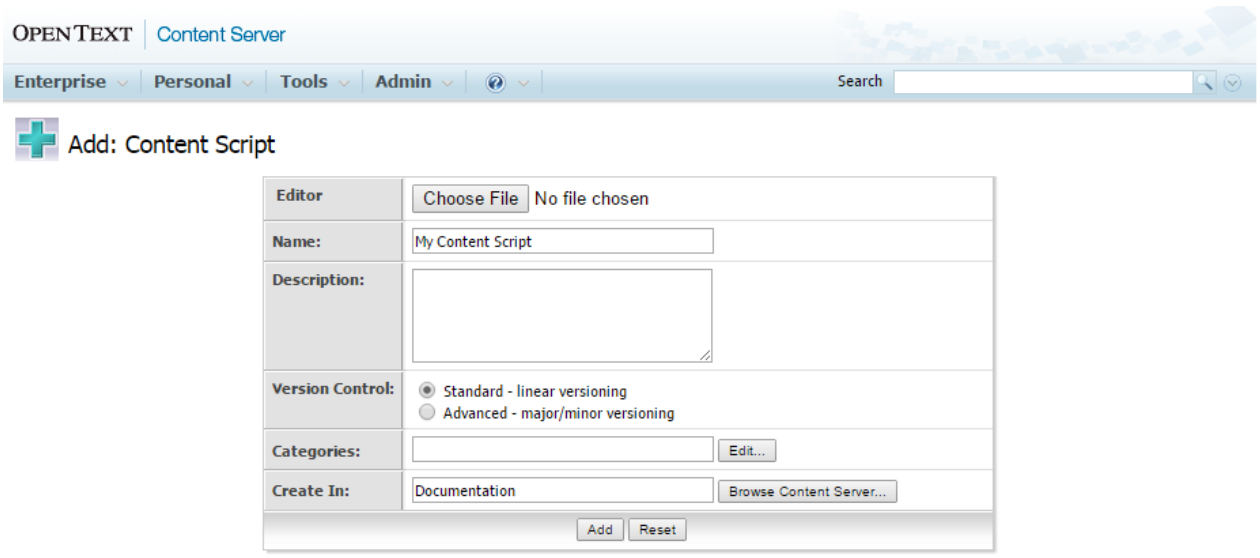
Being standard objects, Content Scripts comply with Content Server **permissions** model. Make sure you assigned the proper permissions to your scripts.

Upon creation, the object can be edited with the web-based IDE selecting the **'Editor'** function in the object function menu. The function is also available as a promoted function.

Creating a Content Script¶

To create a new Content Script object you can leverage the standard **add item menu**:





Object's properties¶

This section covers the following topics:

- Content Script Static variables
- Scheduling a Content Script
- Running a Content Script as a different user
- Changing the default GUI icon for a Content Script object

Static variables¶

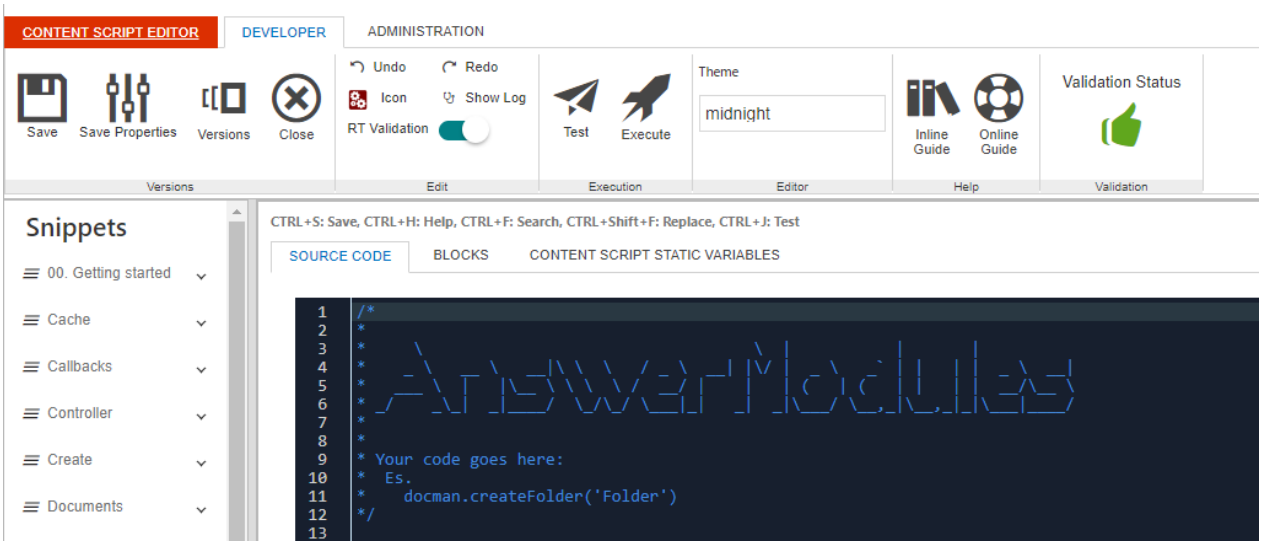
For every Content Script, it is possible to define a set of static, precompiled variables whose values will be available when the script is executed.

The framework supports the definition of these variables by means of a second script, whose outcome is the data map containing the values. For performance reasons, this second script is executed only when it changes (or when execution is explicitly forced by an editor), and the results are stored as part of the script object.

One of the reasons for having a script to define a static variable (instead of explicitly setting the value of the variable itself) is code portability: instead of defining the value of a variable, it is possible to define a rule to calculate that value. A typical example would be the Object ID of an object located in a specific position in Content Server: in case the code is moved to a different environment, the ID would be recalculated automatically.

Static variables are accessible within the Content Script through the **csvars** object.

> Each Content Script object has its own csvars constants. In complex applications, that include multiple Content Script objects, it is often useful to have all constants defined in one single file. This can be done by creating a Content Script dedicated to be the “constants” script, that will be run by the single scripts in the application to load the variable values in the context.



Scheduling¶

Content Script supports the automatic execution of scripts through its internal **scheduler**.

The Content Script **scheduling utility** is available from within the **Specific > Advanced Settings** tab or from the Content Script Editor in the **Administration** tab (if visible). The utility allows to schedule the automatic execution of Content Scripts (that is, without the need for a user to trigger the execution explicitly).

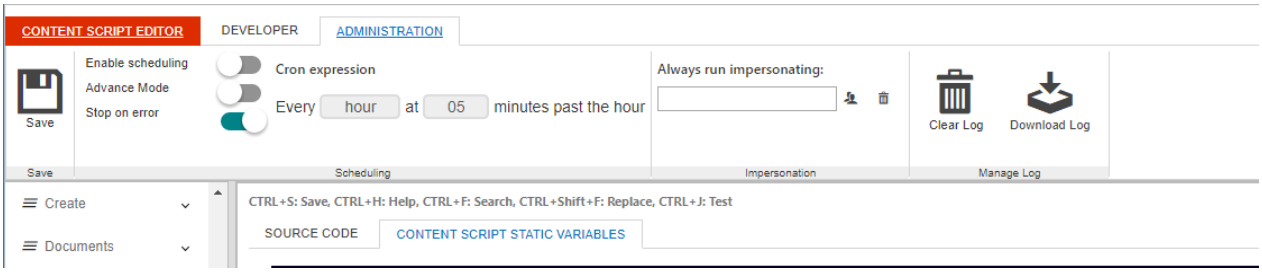
The scheduling is configured by means of a **cron expression**. A cron expression is a string comprising a set of fields separated by spaces, and identifies a set of times.

Cron expressions are powerful but can also be quite complex. For this reason, a simplified configurator with drop-down menus can be used to create the desired cron expression.

Skilled users can always flag the **“Advanced Mode”** checkbox to disable the configurator and compose their own expressions.

Once ready, the scheduler can be enabled by flagging the **“Enable Scheduling”** checkbox.

It is possible to stop a script from being rescheduled in case of execution errors. To do so, simply flag the **“Stop on Error”** checkbox.



Where is the log ?

Content Script scheduling takes advantage of the Content Server's Distributed Agent framework. While normally executing a script will cause it to be run in the current front end server, a scheduled script could actually be executed on any server on which Distributed Agents are activated.

Impersonate¶

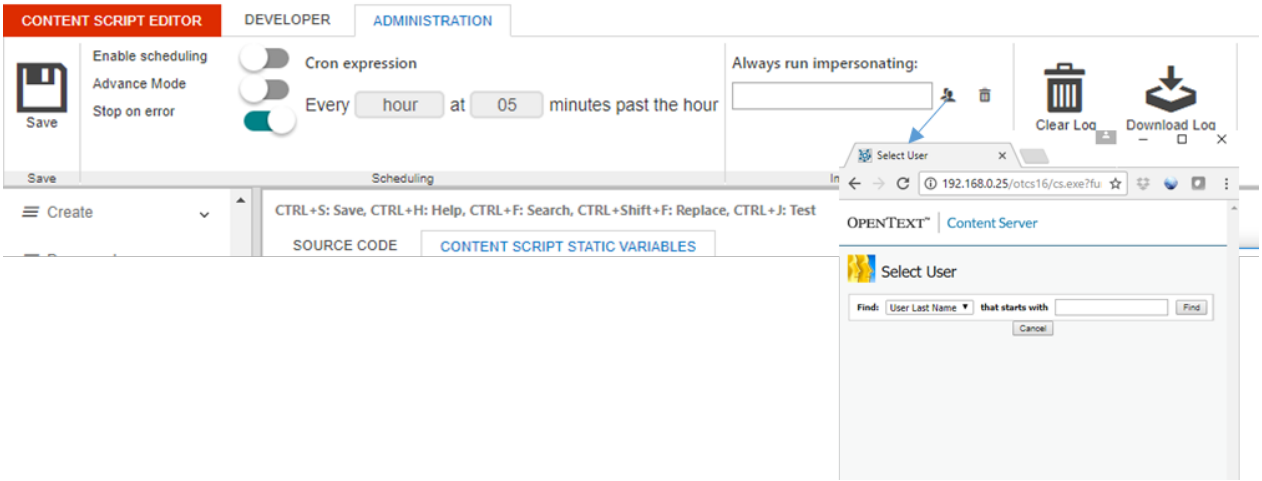
Content Script supports the execution of a script **impersonating** specific users.

This configuration applies for both:

- scripts explicitly executed by users
- scripts executed by the system (scheduled, workflow steps, callbacks, etc...)

The **"Run As"** configuration panel is accessible within the **Specific > Advanced Settings** tab or from the Content Script Editor in the **Administration** tab (if visible).

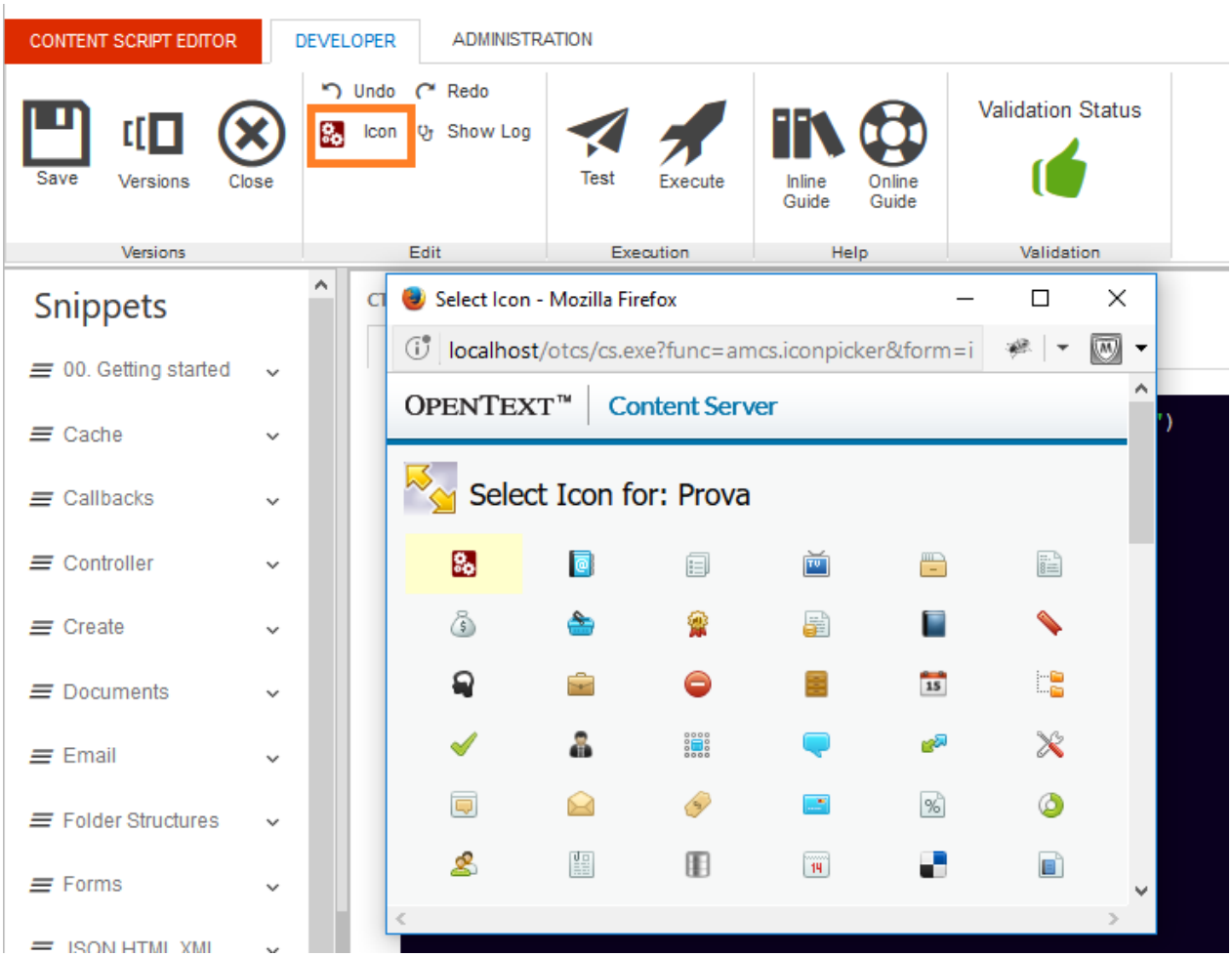
In order to be able to perform a "**Run As**" configuration, a user must have impersonation privileges.



Icon Selection¶

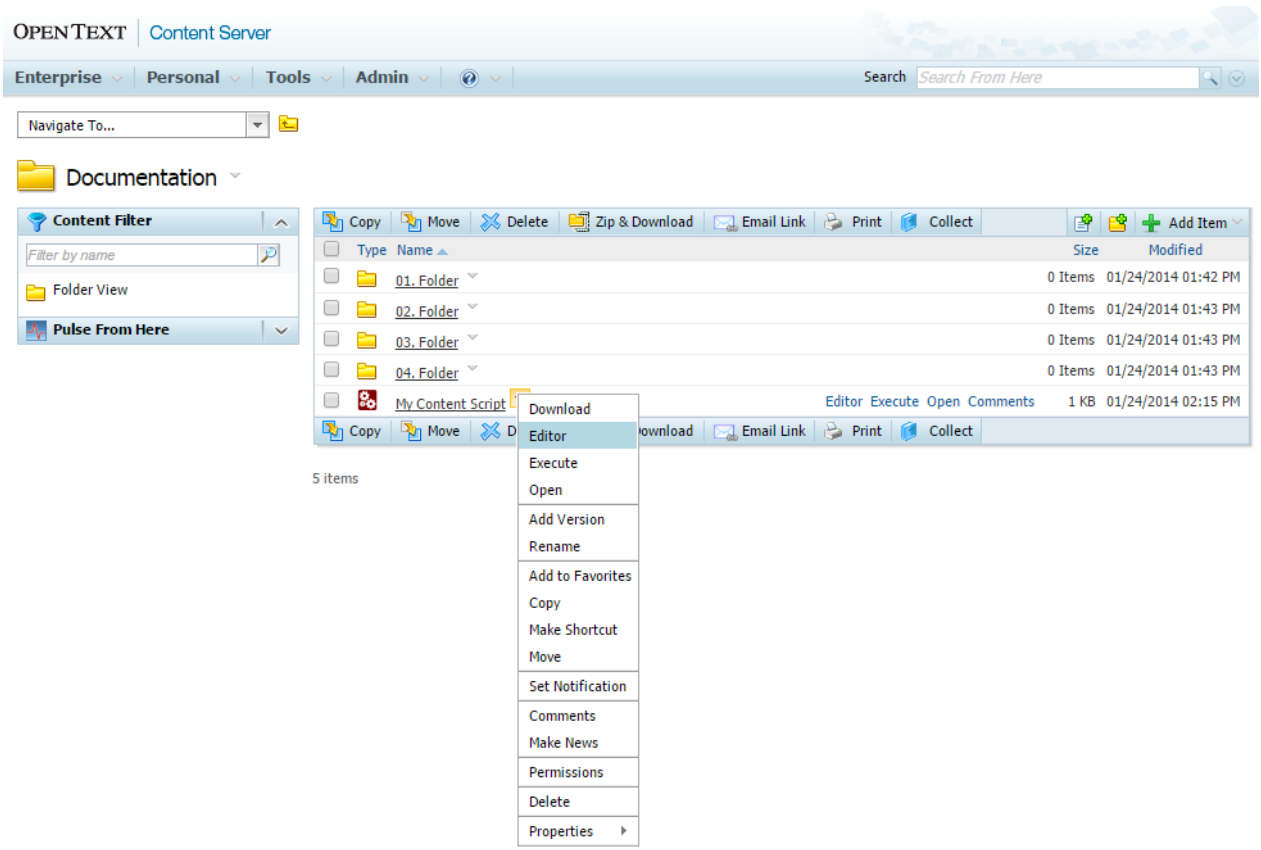
Given the flexible nature of Content Script objects (both in terms of behaviour and execution outcome) it is often useful to be able to distinguish them at-a-glance. One way is to customize the default **icon** used by Content Server for the object.

The desired icon can be selected by clicking on the icon button on the Content Script's Developer tab and selecting a specific icon within a set of available icons.

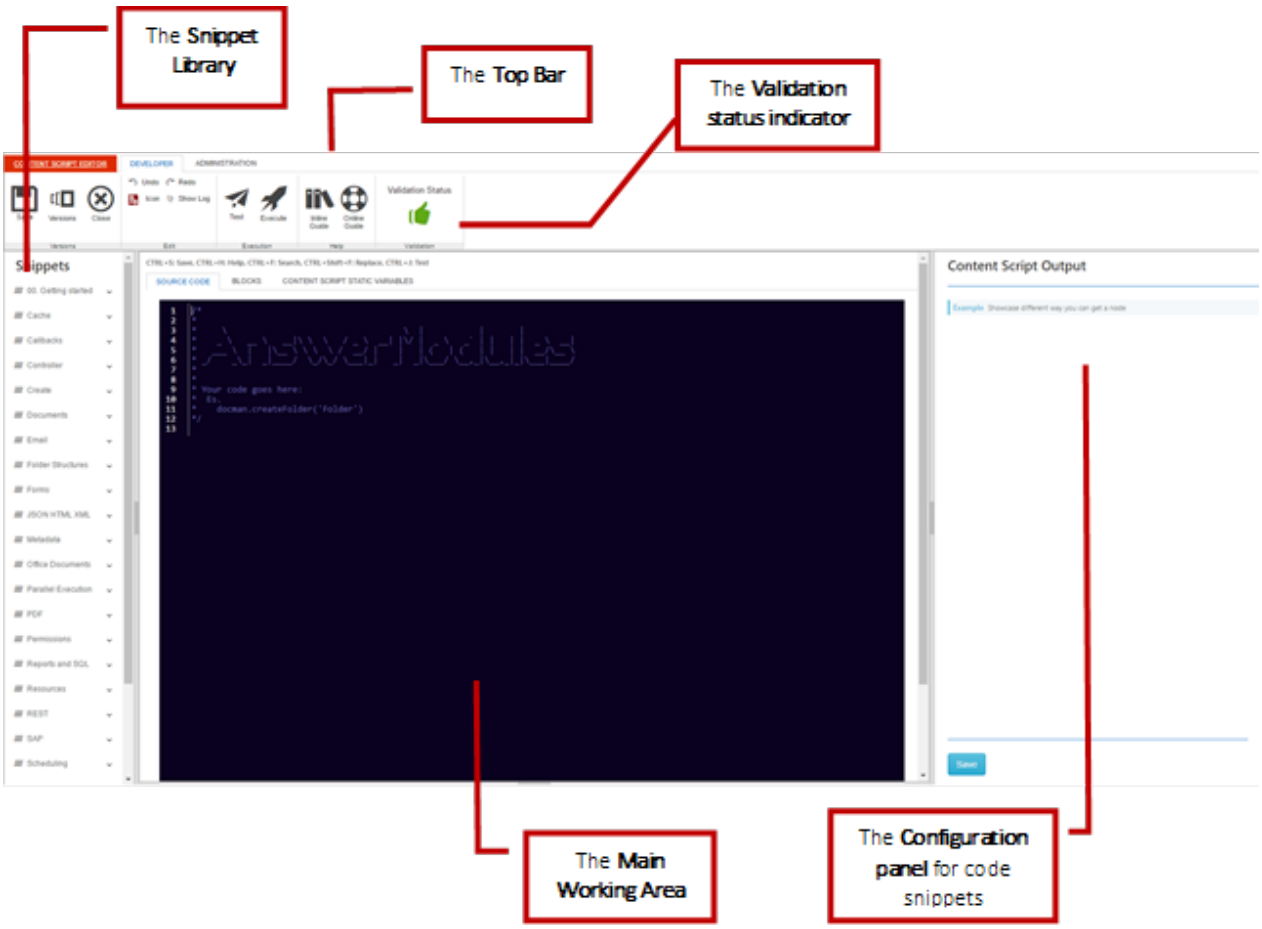


Editor

Content Script objects can be edited with the dedicated web-based IDE selecting the **'Editor'** function in the object function menu. The function is also available as a promoted function.



The web-based IDE (Integrated Development Environment) for Content Script appears as follows:

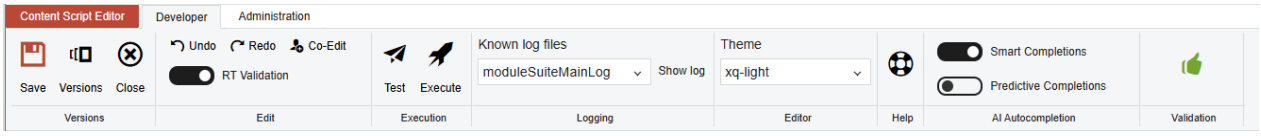


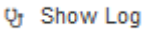
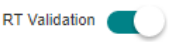
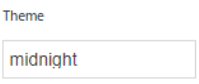
Shortcuts¶

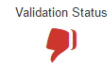
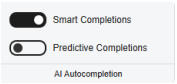
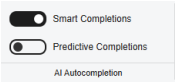
The following keyboard shortcuts are available while using the editor:

Shortcut	Description
Ctrl + S	Save the current script (add a new version)
Ctrl + H	Toggle the online Help window
Ctrl + F	Open the ‘Search’ tools panel
Ctrl + Shift + F	Open the ‘Search and Replace’ tools panel
Ctrl + Space	Show the code autocompletion hints
Ctrl + J	Trigger the execution in the test frame
Ctrl + P	Inject the full path of the selected node in the Content Script editor

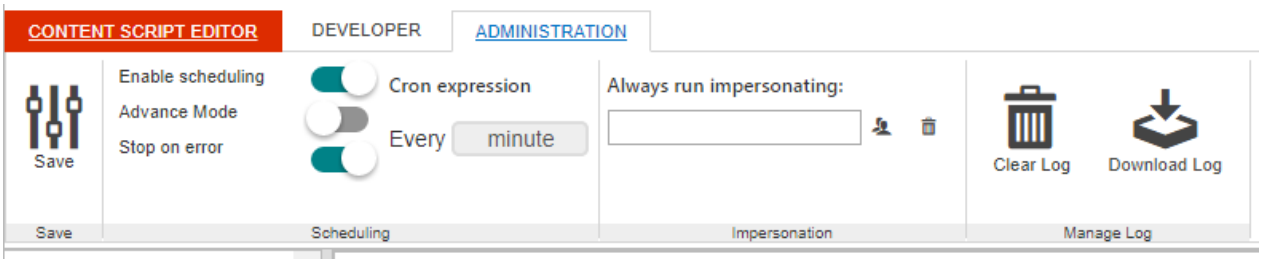
Top Bar controls (DEVELOPER)¶

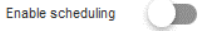
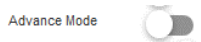


Command	**Description**
Versions	
	**Save** the script (adds a new version)
	**Save** the content script properties (i.e. the icon) as well as the static variables (does not add a new version)
	Open the object's **Versions** tab
	Close the Content Script Editor
Edit	
	**Erases** the last change done
	Opposite of Undo
	Change the script's associated icon
	Display the last 200 lines of the ModuleSuite'smaster log file
	Disable the script's real-time validation
Execution	
	**Run** the script in the current window (CTRL + J)
	**Save** the script and **run** it, showing the result in the editor's bottom panel
Comparison	
	**Toggle comparison**: toggles the comparison of the current version of the script with the selected.
Editor	
	**Theme**:Changes the theme applied to **all** the RPA embedded editors
Help	
	Access the module's online **guide** and the **support portal**

Command	**Description**
Validation	
	**Red label**: The script failed the validation and most likely will **fail** to compile
	**Green label:** The script is **well-formed**
	**Smart Completion:** Activate smart completion to enhance code suggestions
	**Predictive Completions:** When enabled, the editor automatically computes a code suggestion at the current cursor position, even without an explicit request from the user, in order to speed up completion generation. (default: disabled)

Top Bar controls (ADMINISTRATOR)¶

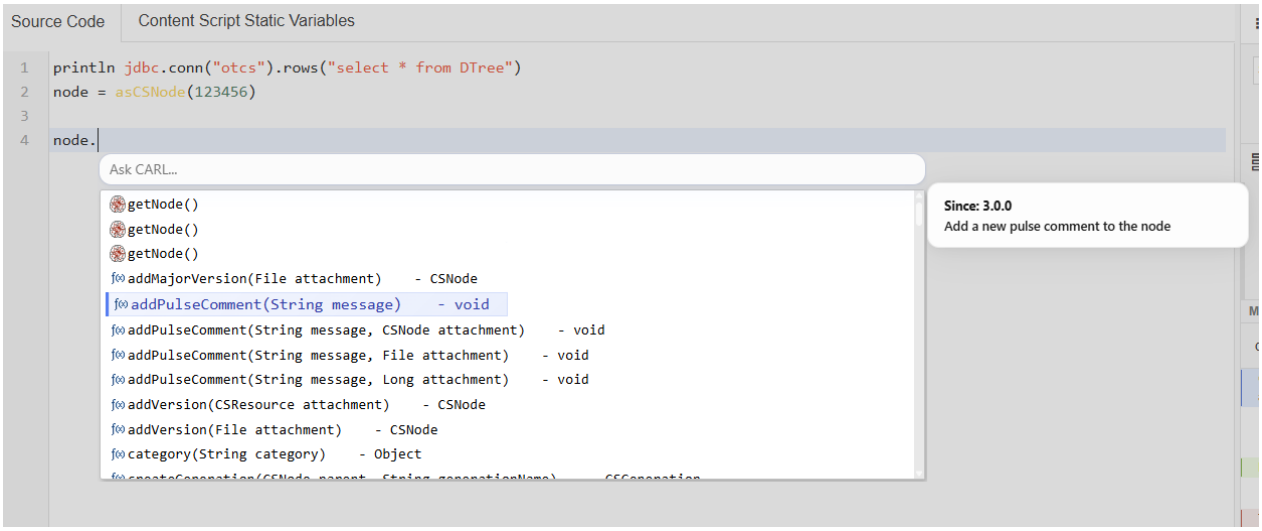


Command	**Description**
Versions	
	**Save** the content script properties (i.e. the icon) as well as the static variables (does not add a new version)
Scheduling	
	**Toggle** script scheduling
	**Toggle** script advance scheduling mode
	**Toggle** re-scheduling abortion on script's execution error
Impersonation	
	**Select** the user that will be always used to run the script
	**Clear** impersonation setting
Manage Log	

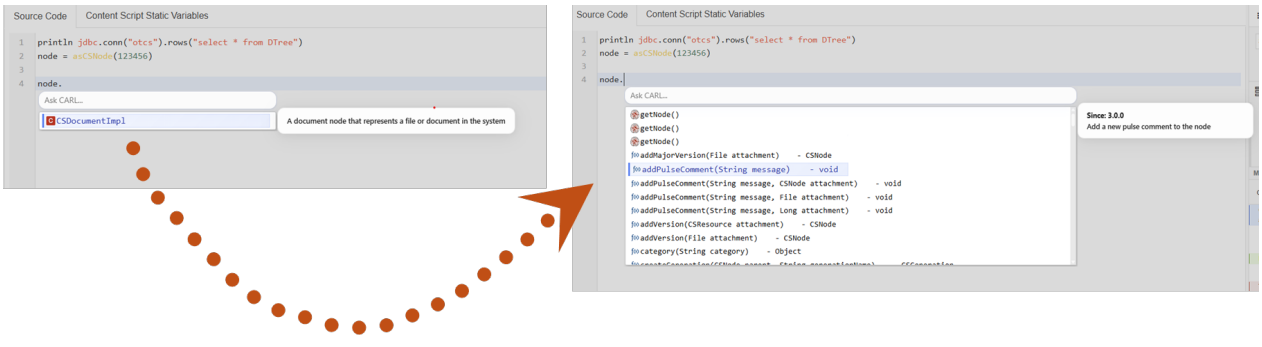
Command	**Description**
Clear Log	**Trigger** ModuleSuite's master log rotation
Download Log	**Trigger** ModuleSuite's master log download

Auto-completion¶

The Content Script Editor features a code completion assistant functionality. While typing use the **ctrl + space** key combination to retrieve the suggested values.



In some cases the Content Script's inference engine might not be capable of determining the actual type of the expression you are trying to auto-complete. In these cases the auto-complete feature will prompt you to firstly specify the type against which the auto-completion should be performed and then will switch to the standard behaviour.



If the actual type (class) of your expression is not listed among the results you can still specify the fully qualified class name to autocomplete against that class: e.g. (java.lang.String)

List of the most common API objects returned by Content Script APIs

| Content Script API Objects ||| |----------|----------|----------|----------| | ACSBrowseViewRowProvider | CSMemberImpl | CSRMRecordTemplate | | AMBWFWidgetsLib |

CSMemberPrivilegesImpl | CSRMRecordTraits | | AdlibJobResult | CSMemberRightImpl | CSRMUserFunctions | | CSANSTemplateFolderImpl | CSMenu | CSRMXReference | | CSAssignmentImpl | CSMenuItem | CSReportImpl | | CSAttachmentImpl | CSMilestoneImpl | CSReportResultImpl | | CSBeautifulWebFormViewImpl | CSMilestoneInfoImpl | CSResourceImpl | | CSBrowseViewAddItemButton | CSNewsBuilderImpl | CSScriptImpl | | CSBrowseViewColumn | CSNewsImpl | CSSearchQueryBuilderImpl | | CSBrowseViewMultiItemButton | CSNodeAuditDataPageImpl | CSSearchResultImpl | | CSBrowseViewRow | CSNodeAuditRecordImpl | CSSetAttributeImpl | | CSCategoryFolderImpl | CSNodeImpl | CSShortcutImpl | | CSCategoryImpl | CSNodePageImpl | CSSpreadsheet | | CSCategoryTemplateImpl | CSNodeResultImpl | CSSubMenu | | CSChannelImpl | CSNodeRightImpl | CSTaskBuilderImpl | | CSCollectionImpl | CSNodeRightsImpl | CSTaskGroupImpl | | CSCompoundDocImpl | CSPDFFormField | CSTaskGroupInfoImpl | | CSCompoundDocReleaseImpl | CSProjectImpl | CSTaskImpl | | CSDiscussionImpl | CSProjectInfoImpl | CSTaskInfoImpl | | CSDiscussionItemImpl | CSProjectPartecipantsImpl | CSTaskListImpl | | CSDocumentImpl | CSProjectRoleUpdateInfoImpl | CSTaskListInfoImpl | | CSEmailImpl | CSRMClassification | CSUnreadInfoImpl | | CSEmailMessage | CSRMClassificationTypes | CSUrlImpl | | CSExportOptionsImpl | CSRMField | CSUserImpl | | CSFTPFile | CSRMFieldsInfo | CSVersionImpl | | CSFolderImpl | CSRMHold | CSVirtualFolderImpl | | CSFormImpl | CSRMHoldDistribution | CSWebReportImpl | | CSFormTemplateDefinitionImpl| CSRMHoldDoc | CSWordDoc | | CSFormTemplateImpl | CSRMHoldPage | CSWorkPackageImpl | | CSGenerationImpl | CSRMProvenance | CSWorkflowAssignedTaskImpl | | CSGroupImpl | CSRMRSIRetention | CSWorkflowAttachmentsImpl | | CSImportOptionsImpl | CSRMRecord | CSWorkflowAttributesImpl | |CSWorkflowAuditRecordImpl | CSWorkflowCommentsImpl |CSWorkflowFormDataImpl | |CSWorkflowInstanceImpl |CSWorkflowMapImpl | CSWorkflowQueryBuilderImpl | |CSWorkflowSearchHandleImpl |CSWorkflowStartDataImpl |CSWorkflowFormsImpl | | CSWorkflowTaskActionsImpl |CSWorkflowTaskCommentImpl |CSWorkflowTaskDetailsImpl | |CSWorkflowTaskImpl | CSWorksheet |FTPConfigProfile | |FieldInfo |Form |GCSAdlibJob | |GCSCategory |GCSTableOfContents | GCSWatermark | |LDAPConnection |NodeListRowProvider |PDFOverlayText | |PDFWaterMark |SQLQueryRowProvider | SampleContextAwareObject | |SampleObject |SearchResultRowProvider |SinglePageRowProvider |

AI Autocompletion¶

The AI Autocompletion feature provides intelligent, context-aware code suggestions directly within the Script Editor. When enabled, it calculates a variable number of potential code completions for the code immediately to the left of the cursor.

To generate relevant completions, the system analyzes both the full set of available Content Script APIs and the current context of the script being edited. This is considered an *experimental* feature, and must be explicitly enabled via the **Smart Completion** switch in the editor options. As a prerequisite, the C.A.R.L. integration must be properly configured and enabled on your system.

In addition to manual suggestions, there is a **Predictive Completion** switch. If this is enabled, the editor will automatically compute possible code completions based on the cursor's current position, even if the user hasn't explicitly requested them. This allows the system to keep a background "cache" of likely completions ready for the user, making the experience more seamless and responsive.

Note

- **Smart Completion** must be enabled for AI-driven code suggestion to work, and requires C.A.R.L. to be active and properly configured.
- **Predictive Completion** enhances responsiveness by precomputing possible completions in the background and storing them in a cache.

These options can be toggled in the Script Editor UI, giving you control over the level of AI assistance you want to use during script development.

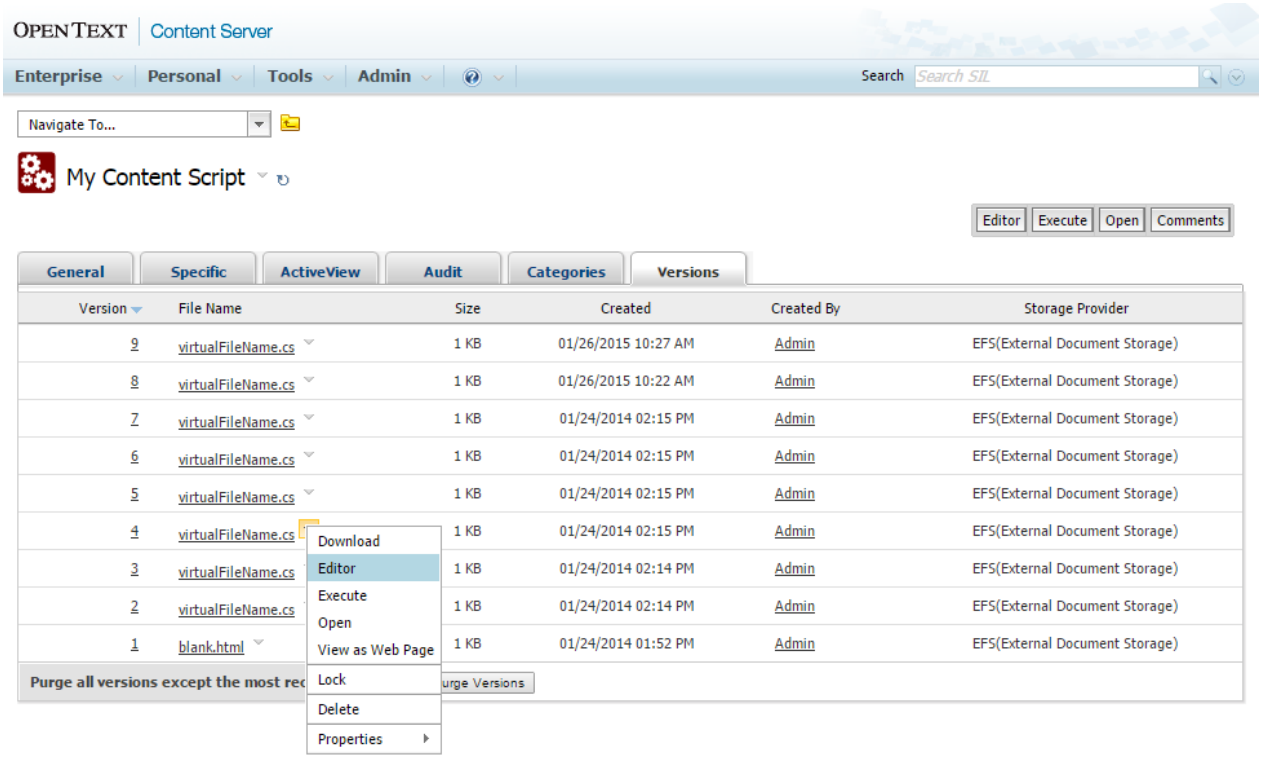
Code Validation¶

Every time a change is made to the script, a code validator attempts to check the formal correctness of the code. A **validation status** icon placed on the bottom right side of the working area will show the result of the validation. Code that fails the validation status check will most likely contain formal errors and will fail to compile correctly, if executed.

Versions tab¶

Content Script objects are subject to versioning on Content Server, just like any other document-class object. Every time the Content Script is saved in the IDE, a new version is created.

Older versions of the Script can be opened in the Script Editor for editing. If saved, a new (latest) version will be created.

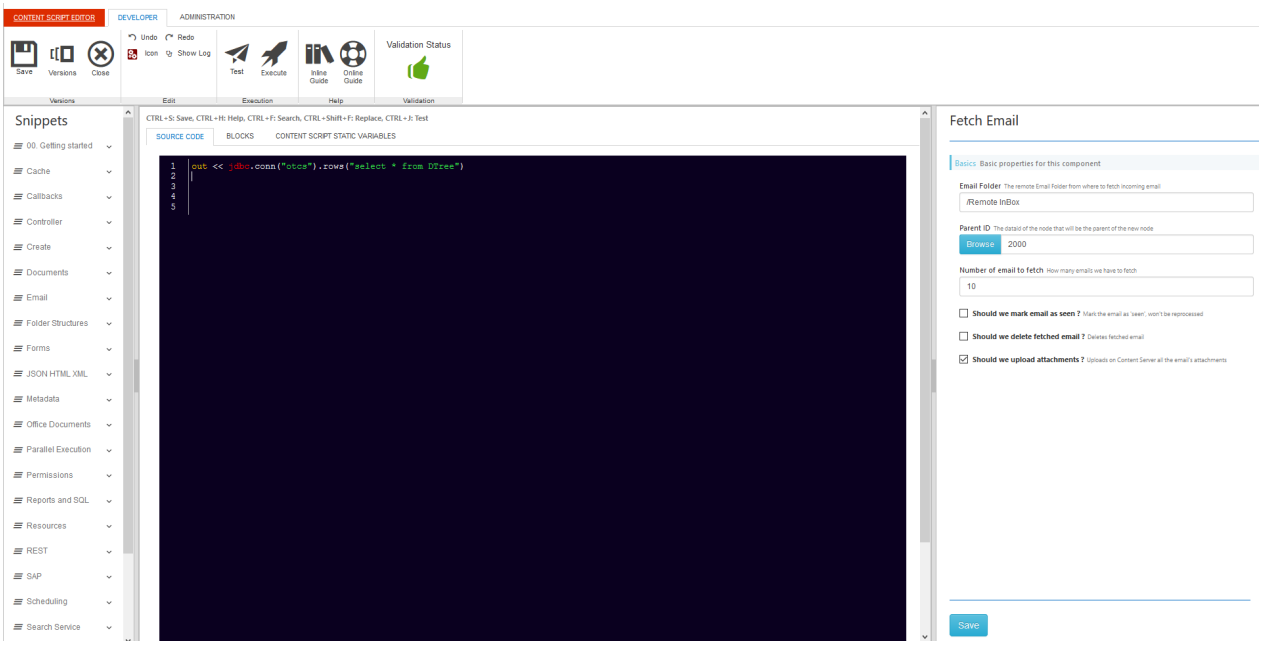


Code Snippet library¶

In order to simplify the creation of new scripts, a library of pre-configured **ready-to-use code snippets** is available in the Script Editor.

Snippets are grouped in **families** of objects with similar features. In order to use a snippet in Content Script:

1. Navigate the library until you find the suitable snippet
2. Place the cursor in the Working Area location where you wish to place the code
3. Click on the snippet to open the Configuration Panel
4. The code snippet could contain place-holders for some configuration variables (for example, in case of a “create document” snippet, a configurable option could be the target container where to create the document.) In this case, configuring the variables as required.
5. Click **Save**. The resulting code will be placed at the location of the cursor in the working area.



> once the code is placed in the working area, it can be further modified as necessary.

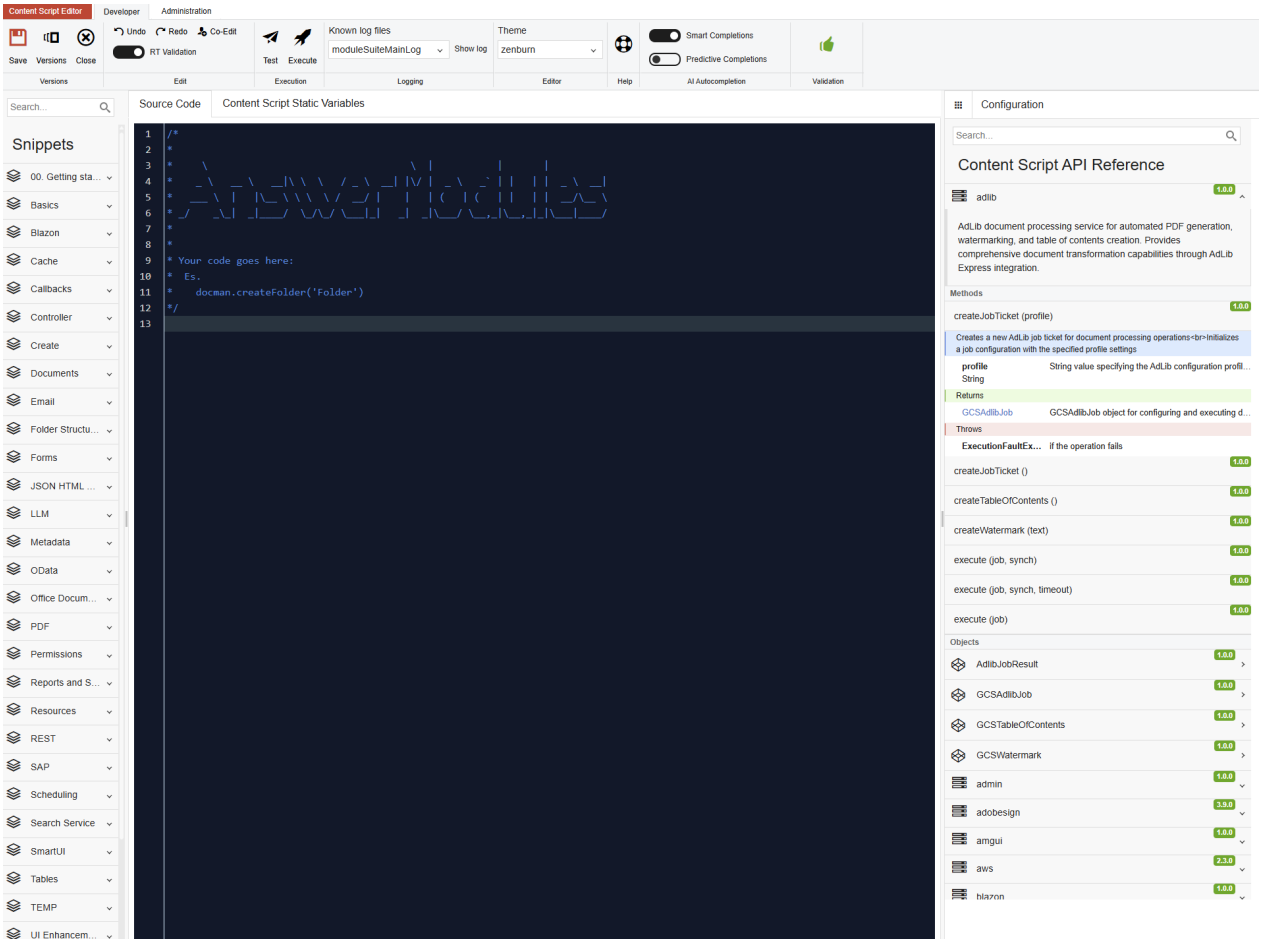
Online Help¶

The Content Script IDE features two different online help guides:

- The complete API Reference (accessible with the **Ctrl + H** shortcut)
- The Content Script Module Help (accessible through the standard Content Server Help, or through the “**Help**” button in the Top Bar of the IDE)

The **Content Script API Reference** can be toggled in a navigable panel on the right side of the screen and describes the programming interfaces of all objects and services that are available

in the Content Script context. A more thorough description of the available APIs is presented in the following sections.



Language basics

Content Script is a **Domain-Specific Programming Language (DSL)** for OpenText Content Server.

The language is based on **Oscript** and exposes a **Groovy** interface to developers. **Groovy** (http://www.groovy-lang.org/ *(http://www.groovy-lang.org/)*) is a widespread dynamic language for the Java Virtual Machine, particularly indicated for the creation of DSLs.

Content Script language syntax is fully compatible with Groovy.

Under the hood, a mix of **Oscript** and **Java** features allow for a deep integration with Content Server functionalities, as well as for an extreme ease of **integration with external systems**.

The following sections are meant to be an introduction to the language.

Statements¶

The definition of variables can be either generic or restricted to a specific type. Assigning a value to a variable that does not match its type will force the engine to attempt to cast its value to the given type. In case no conversions can be done, it will result in an error.

```
// Defining a local variable can be done either by
// 1) declaring explicitly its type
// 2) using the "def"
int anInt = 1

String aString = "text"

def anObject = "anything"
anObject = 123
```

With String variables, a few useful tricks are available:

```
// Strings can be defined both with quotes ('') or double quotes ("")
String aString = "text"
String anotherString = 'text'

// Selecting the alternative "" or '' can be useful if quotes are present in the string content
String aQuote = "this is a quote: 'My words...' "
String anotherQuote = 'this is a quote: "My words..." '

// using triple """ allows to span across multiple lines for string
// definition. Useful for readable SQL queries, for example..
String multilineString = """SELECT *
                            FROM DTREE
                            WHERE DATAID = 2000"""

Lists and Maps can be defined very easily
def aList = ["firstElement", "secondElement"]

def aMap = [firstKey:"firstValue", secondKey:"secondValue"]

// statements can span across multiple lines
def multilineDefinition = ["firstElement",
                            "secondElement"]

// collections can contain different kinds of elements
def aMapWithStringsAndInts = [first:"one", second:2
```

Basic Control Structures¶

Below are the basic structures for flow control and iteration

- **if – else** statement
- **if – else if – else** statement
- **inline if** statement
- **switch** statement

- while loop
- for loop

Flow control: if – else¶

```
if(a == b){
    //do something
} else {
    //do something else
}
```

Flow control: if - else if - else¶

```
if(a == b){
    // do the first thing
} else if(c == d){
    // do a second thing
} else {
    // do something else
}
```

Flow control: inline if - else¶

```
a = (b == c) ? "c is equal to b" : "c is different from b"
```

Flow control: switch¶

```
switch ( a ) {
    case "a":
        result = "string value"
        break

    case [1, 2, 3, 'b', 'c']:
        result = "a mixed list of elements"
        break

    case 1..10:
        result = "a range"
        break

    case Integer:
        result = "is an Integer"
        break

    case Number:
        result = "is a Number"
        break

    default:
        result = "default"
}
```

Looping: while¶

```
def a = 0

while (a++ < 10){
    // do something ten times
}

def b = 10

while ( b-- > 0 ) {
   // do something ten times
}
```

Looping: for¶

```
// Standard Java loop
for (int i = 0; i < 5; i++) {

}

// range loop
for ( index in 0..100 ) {
    // do something
}

// list or array loop
for ( index in [0, 10, 20, 40, 100] ) {
    // do something 5 times
}


// map looping
def aMap = ['first':1, 'second':2, 'third':3]

for ( entry in aMap ) {
    // do something for each entry (the values can be accessed and used)
    entry.value
}
```

Operators¶

All Groovy operators can be used in Content Scripts:

Operator Name	Symbol	Description
Spaceship	<=>	Useful in comparisons, returns -1 if left is smaller 0 if == to right or 1 if greater than the right
Regex find	=~	Find with a regular expression
Regex match	==~	Get a match via a regex
Java Field Override	.@	Can be used to override generated properties to provide access to a field
Spread	*.	Used to invoke an action on all items of an aggregate object

Operator Name	Symbol	Description
Spread Java Field	*.@	Combination of the above two
Method Reference	.&	Get a reference to a method, can be useful for creating closures from methods
asType Operator	as	Used for groovy casting, coercing one type to another.
Membership Operator	in	Can be used as replacement for collection.contains()
Identity Operator	is	Identity check. Since == is overridden in Groovy with the meaning of equality we need some fallback to check for object identity.
Safe Navigation	?.	returns nulls instead of throwing NullPointerExceptions
Elvis Operator	?:	Shorter ternary operator

Methods and Service Parameters¶

Methods on objects can be called using the dot "." followed by the method signature and parameter clause.

```
out << template.evaluateTemplate("""
#@csform(false, "Submit")
    <label for="myFile">File to be uploaded</label>
    <input type="file" name="myFile"  />
#end
""")

if(params.myFile && params.myFile.filelength){
    def parentNode = docman.createFolder("MyFolder")
    def file = new File(params.myFile)
    if(file && file.canRead()){
        docman.createDocument( parentNode, params.myFile_filename, file, "", false, parentNode)
        //Redirect after submit
        redirect "${url}/open/${self.ID}"
    }
}
```

Methods can be called omitting the parenthesis in the parameter clause, given that (a) there is no ambiguity and (b) the method signature has at least one parameter.

```
// In certain cases, parenthesis can be omitted
docman.createFolder "MyFolder"
```

Properties and Fields¶

Properties and public fields of objects can be accessed using the dot "." followed by the property or field name.

```
def folder = docman.createFolder("myFolder")

// Accessing an object property
def me = folder.createdBy
```

A safe syntax to navigate through fields is available in Groovy by adding a "?" before the dot. In this case, the chain will be interrupted if one of the intermediate values is undefined, avoiding an exception to be raised.

```
// Safe field access (no exception raised if folder is NULL)
def me = folder?.createdBy
```

Comments¶

```
// Comments are available as single line // and multiline /* */

def a = 1 // A comment can close a line

/* Or span
over multiple
lines */
```

Closures¶

Content Script inherits from Groovy the concept of Closures. A closure is an open, anonymous, block of code that can take arguments, return a value and that can be assigned to a variable.

```
// Define a closure and assign it to a variable
def addNumbers =  {  def num1 , def num2 -> //Arguments
    return (num1 as int)+(num2 as int)
}

out << "Calling the addNumbers closure:${addNumbers(4, "5")} <br/>"

addNumbers = { String... arguments -> // Variable number of arguments (MUST be the last parameter)
    def total =0
    arguments.each{total+=(it as int)}
    return total
}

out << "Calling the addNumbers closure:${addNumbers("1", "2","3")} <br/>"

def createNewFolder = {  String name,  def parentNode = docman.getEnterpriseWS() ->
    docman.createFolder(parentNode, "name" )
}

def node = createNewFolder( new Date().format("yyyyMMddHHss"))
out << "Calling the createNewFolder with One arguments:${node.ID} <br/>"

def newNode = createNewFolder( new Date().format("yyyyMMddHHss"), node)
out << "Calling the createNewFolder with Two arguments:${newNode.ID} <br/>"
```

Content Script programming valuable resources¶

A number of resources can be extremely useful to the Content Script developer at different times. A few of the most important ones are:

Online help

The Content Script Module features an online guide that covers the basic language syntax and functionalities. It also contains quick references to context variables and methods.

Code Snippet Library

When using the Content Script Editor, a library of ready-to-use code snippets is available to bootstrap new scripts without having to start from scratch. The library includes usage examples and code templates for many common use cases, and can be easily extended by the developer.

Groovy reference guide

The Apache Groovy language is supported by a wide community of adopters worldwide. Groovy is supported by the Apache Software Foundation: a significant amount of documentation and examples are available online.

http://www.groovy-lang.org/ *(http://www.groovy-lang.org/)*

Velocity reference guide

The Apache Velocity engine powers the templating features in Content Script. Velocity is supported by the Apache Software Foundation: lots of documentation and examples can be found throughout the web and on the project's website.

http://velocity.apache.org/ *(http://velocity.apache.org/)*

Writing and executing scripts

Content Script scripts are "document" class objects stored on Content Server. The primary usage for a script is its execution. When you "execute" a script, you are basically programmatically invoking a series of APIs that perform actions over Content Server's or other systems' data. In the following paragraphs, we are going to analyze all the Content Script architecture's elements and components that play a role in turning a textual file into an actionable object.

As said, scripts are persisted as "documents" on Content Server. Whenever you execute a script a component named Script Manager retrieves the script's last version and, either compiles it (and caches the compiled version) or loads a pre-compiled version of it for execution. Scripts'

execution is managed by another component named Content Script Engine. The Content Script Engine executes the script's code against the provided execution context (the execution context is the "container" through which the script's code can access the Content Script's services, environment variables, support variables, database, etc..). The internals of both the Script Manager and the Script Engine are not relevant for the purpose of this manual and won't be discussed.

API Services¶

Content Script API Service¶

Content Script APIs are organized in classes denominated **services**. Each Content Script API service acts as a container for a set of *homogeneous* APIs (API releated to the same kind of objects or features). Content Script APIs can be extended creating and registering new services *(/working/contentscript/sdk/#create-a-custom-service)*.

Content Script APIs are, in their most essential form, the methods exposed by the service classes. In order to be recognize as a Content Script API a service class method must be decoretad with the `@ContentScriptAPIMethod` annotation.

> **Content Script API Services Interfaces**
>
> When working with Content Script APIs developers program against interfaces. As a matter of fact all Content Script API services and objects implement one or more interfaces. Implementation classes can be easily distinguished from their interfaces because their name ends with the "Impl" suffix.

Content Script API Objects¶

Content Script APIs return or accept, as parameters, objects representing OTCS objects or features. In Content Script, these objects are referred to as **Content Script API objects**. Content Script API objects are *active* information containers. We define them *active* because they expose APIs designed to manipulate the information stored in themselves.

In order to be recognize as a Content Script API Object a class must be decoretad with the `@ContentScriptAPIObject` annotation.

When the script Execution Context is initialized by the Content Script engine, all registered API services are injected into it. These **services** allow a Content Script to perform operations on Content Server, to use internal utilities (such as PDF manipulation utilities or the templating service), to access external systems and services, etc.

Here after are some of the main services that are currently available as part of Content Script APIs.

API Service Name	Description
Base API	Base API is constituted by methods and properties that are exposed directly by each script. Some of the most important API are: logging, redirection (used to redirect users navigation through a server side redirection i.e. http code 302, outputting HTML, XML, JSON and Files)
Core Services	
admin	The admin service allows to programmatically perform administrative tasks. With the admin service is it possible, among other things, to: perform XML import/export operations, programmatically schedule/unscheduled Content Script executions
collab	The collab service is the main access point to the Content Server collaborative functionalities. With collab service is it possible, among other things, to: create and manage projects, tasks and milestones, create and manage discussions, list and manage users' assignments
distagent	The distagent service provides functionality for: scheduling and unscheduling of scripts, comprehensive configuration management. distagent supports the definition of **MapReduce type of Jobs** (referred to as "Chain Jobs"), which allows users to configure behaviors for each phase of the job:**Split**: Define how to partition the data. **Map**: Specify the processing for each partition. **Reduce**: Aggregate the results from the map phase. **Finalize**: Conclude the job with any post-processing steps required.
docman	The docman service is the main access point to the Content Server Document Management functionalities. With docman service it is possible, among other things, to: create and manipulate documents and containers, access and modify meta-data, access and modify object permissions, access volumes, perform database queries, manipulate renditions and custom views, run reports, consume OScript request handlers, programmatically import/export content through Content Server native XML import/export feature
mail	The mail service allows to programmatically create/send and receive emails from scripts. With the mail service is it possible, among other things, to: create and send email message through multiple mailboxes, scan mailboxes and retrieve incoming messages and attachments, create email messages (both html and text messages are supported) with custom templates, send email to internal users and groups, attach files and Content Server documents to emails, configure multiple email service profiles to use different IMAP/SMTP configuration at the same time
search	The search service allows to programmatically search over Content Server's repository. With the search service is it possible, among other things, to: easily build/execute complex search queries programmatically, easily build/

API Service Name	Description
	execute query based on categories attributes, retrieve search result with or without pagination
users	The users service is the main collector for all APIs related to Content Server users and groups. With users service is it possible, among other things, to: create/modify/delete users and groups, impersonate different users, access and modify user privileges, perform member searches
template	The template service can come in handy anytime you have to dynamically create documents. With the template service is it possible, among other things, to: evaluate documents and plain text strings as templates, replace place holders and interpret template-expressions
workflow	The workflow service allows to programmatically manipulate workflows. With the workflow service is it possible, among other things, to: start, stop, suspend, resume, delete workflows, access and manipulate workflow and task data, accept, complete, reassign workflow tasks, perform searches within workflows and tasks, change workflows' and steps' title
Extensions	
adlib	The adlib service allows to programmatically drive the AdLib rendition engine. With adlib service it is possible, among other things, to: create jobs for AdLib PDF Express Engine and fetch renditions results
amgui	The `amgui` service is designed to control the visibility of elements within the Classic UI upon the execution of a content script, and it facilitates the retrieval of format specifications for certain data types, such as dates. It also handles the data structures inherent to the Classic UI, enabling the creation of custom pages and reports. Furthermore, `amgui` is instrumental in generating SmartView session tickets and provides programmatic access to the form builder for server-side creation of Beautiful Webforms views.
amsui	The `amsui` service is primarily utilized by the Smart Pages module to ascertain which enhancements, such as commands, actions, panels, and columns, should be activated in the SmartView and their respective locations. Additionally, `amsui` is responsible for the server-side rendering of Smart Pages as well as generating Smart View session tickets, which are essential for the integration of Smart View-related widgets within Beautiful WebForms and Smart Pages.
aws	The `aws` service provides an interface for accessing and managing Amazon EC2 instances and S3 storage services. It enables the management of S3 buckets and the orchestration of EC2 resources, streamlining cloud operations within Amazon's extensive infrastructure.
blazon	

API Service Name	Description
	The `blazon` service is designed to facilitate programmatic access to the OpenText Blazon rendition engine, enabling developers to create and manage rendition jobs. This service allows for the programmatic initiation of jobs and their management in both synchronous (waiting for job completion) and asynchronous modes. It grants full access to the Blazon API suite, encompassing a comprehensive range of rendition and transformation capabilities.
cache	The `cache` service provides interaction capabilities with the distributed Memcache service, offering APIs to store and retrieve data applicable to individual users or the entire Content Server population. This service simplifies the integration within Content Script objects, such as `CSNode` nodes, through a well-designed API set that streamlines cache usage in the context of content management operations.
classification	The classification service is the main access point to the Content Server classification features. With classification service is it possible, among other things, to: access, apply, remove classifications from objects
core	The `core` service is tailored to offer simplified access to integrated core services, including 'share' and other essential components. This service streamlines the process of interfacing with the fundamental functionalities of our system, enhancing developer efficiency and system integration.
docbuilder	The `docbuilder` service acts as a wrapper around an enhanced version of the (Groovy Document Builder library *(http://www.craigburke.com/document-builder/)*). This service facilitates the creation of both PDF and Word documents, with the capabilities for generating Word documents being significantly extended beyond those offered by the underlying library.
docx,xlsx	The docx/xlsx services allow to programmatically manipulate Microsoft Office documents. With docx/xlsx services is it possible, among other things, to: create and manipulate Word, PowerPoint and Excel documents, read and write documents' properties
eng	The `eng` service offers access to a comprehensive set of APIs specific to Extended ECM for engineering. This includes specialized functionalities for managing CAD documents, handling transmittals, state flows, distribution matrices, and other engineering-specific processes and data structures.
forms	The forms service is the main access to the Content Server web-forms features. With forms service it is possible, among other things, to: create and modify `form` and `form template` objects, read/modify/delete submitted form records, submit new form records, export/import form records
ftp	

API Service Name	Description
	The ftp service allows to interact with FTP services. With ftp service it is possible, among other things, to: access, read, write files and folders on multiple FTP servers
html	The `html` service provides a convenient set of server-side APIs for processing HTML code. Key features include HTML to XHTML conversion, HTML sanitization, XSS (Cross-Site Scripting) prevention, and HTML to PDF conversion, all achievable without the need for additional software.
jdbc	The `jdbc` service is designed to offer a convenient method for integrating JDBC-enabled data sources. It manages connections to multiple data sources and connection pools to optimize performance. Additionally, this service includes a layer of abstraction related to the specific pooling technology used, enhancing its adaptability and ease of use in various database environments.
ldap	The `ldap` service is dedicated to managing LDAP (Lightweight Directory Access Protocol) connections and operations, streamlining interactions with LDAP servers and simplifying directory management tasks.
llm	The `llm` service is designed to integrate LLM (Large Language Model) based services, specifically tailored for the context of the Module Suite application. It offers a convenient set of APIs for use in scripts and widgets, facilitating seamless integration. Currently, the service supports OpenAI-like APIs but is architecturally agnostic, allowing compatibility with various underlying LLM technologies.
notifications	The `notifications` service is designed to provide programmatic access to the functionalities of the Notification Center. This enables developers to utilize it for notifying users about events related to Module Suite applications, enhancing user engagement and awareness.
oauth	The `oauth` service is specifically designed to simplify the access and usage of resources and services protected by OAuth, streamlining authentication and authorization processes for secure and efficient integration.
odata	The `odata` service is crafted to offer a convenient approach for both consuming and producing REST APIs that are compliant with OData (Open Data Protocol) standards, facilitating seamless interaction with OData-based services. For more information on OData, you can visit the OData official website *(https://www.odata.org/)*.
pdf	The pdf service allows to programmatically manipulate PDF documents. With pdf service is it possible, among other things, to: create and manipulate PDF documents, write in overlay on PDFs, extract PDF pages as images, merge PDFs, add watermarks to PDF documents, add barcodes (mono and bi-dimensional) on PDF pages, remove print/modify permissions from PDF, add

API Service Name	Description
	PDFs in overlay to existing PDFs, extract images from pages or portion of pages, read bar-codes form PDF's pages, remove/insert pages
physobj	The `physobj` service is designed to facilitate the management of Physical Objects nodes, providing tools and functionalities necessary for handling these specific types of nodes effectively.
recman	The `recman` service is intended to provide access to Records Management APIs and services, enabling the integration and utilization of records management functionalities within the application framework.
rend	The rend service allows to programmatically invoke external rendition engines. With rend service it is possible, among other things, to: transform on the fly HTML pages to PDF documents, rend WebForms as PDFs, invoke external services through an "all-purpose" generic rendition api
rmsec	The `rmsec` service is intended to provide access to APIs and services related to Records Management Security and Security Clearance modules, enabling streamlined integration and management of these critical security features.
rtl	The `rtl` (Right-to-Left) service features a set of APIs specifically designed to simplify the creation of user interfaces that support Right-to-Left languages, ensuring ease of use and inclusivity in global applications.
sap	The sap service allows to integrate Content Script with the well known SAP ERP through RFCs. With sap service it is possible, among other things, to: connect to multiple SAP systems through JCO APIs, invoke standard and custom SAP functions to retrieve/update ERP information
sftp	The `sftp` service encompasses a suite of APIs tailored to simplify, optimize, and enhance the efficiency of using SFTP (Secure File Transfer Protocol) services in various applications.
sql	The `sql` service is designed to facilitate access to the platform's underlying database. This service enables querying the database, managing the creation of cursors, wrapping queries in transactions, and more. Additionally, it features methods specifically aimed at simplifying the creation of SQL queries for paginated data access, streamlining interactions with large datasets.
sync	The `sync` service is designed to support the configuration of server-side services and APIs necessary for Syncfusion-based widgets. This includes both Beautiful WebForms and Smart Pages widgets, ensuring seamless integration and functionality within these frameworks.
xecm	The `xecm` service is tailored to support the management and creation of new Extended ECM connectors, as well as Extended ECM-related objects such as

API Service Name	Description
	Business Workspaces. Additionally, this service facilitates integration with the Event Bots center, enhancing the capabilities of Extended ECM environments.
zip	The `zip` service is designed to provide robust functionalities for handling ZIP file operations. This includes creating, extracting, and managing ZIP archives, enabling efficient file compression and decompression within various applications. The service is optimized for ease of use and seamless integration, making it ideal for managing large datasets or grouped files in a compressed format.

APIs evolution

New service APIs are constantly added or updated with every subsequent release of Content Script. Optional APIs are usually available through Content Script Extension Packages, and can be installed separately using the master installer or the extension packages' own installers.

Execution context¶

Upon execution, every Content Script is associated to a Groovy binding. The binding can be seen as a container for Objects that are part of the context in which the script is executed. We make reference to this context as Content Script Execution Context or as Script Binding.

The Script Manager creates the most appropriate execution context on the basis of:

- the script's code;
- the system's current configuration;
- the user context (user's permission, user's roles, etc..)
- the cause that triggered the script's execution (direct invocation, scheduler, callback, etc..)

```
graph LR
  A[Script Source Code] --> B([Script Manager]);
  B --> |Compiles| C{{Script Compiled}};
  B --> | Assemble| D[Execution Context];
  C --> E([Script Engine]);
  D --> E;
  E --> |Executes the Script against the Context and generates| F[Result]
```

The Script Manager initializes the Script Binding before execution, injecting a set of objects, which include:

- API Services

- Request variables
- Support Objects
- Support Variables

Additionally, a set of script utility methods are available in the Content Script (Base API). The methods grant access to short-cuts for commonly used features or can pilot the execution result.

Request variables¶

Request variables are variables injected into the execution context by the Script Manager whenever a script is directly invoked as a result of a user's browser request.

Variable	Description
`params`	A container for the Script's request parameters. It's a non-case sensitive map that provide access to all the parameters passed to the script when executed. In the params map are injected by default also the following variables (where available): • myurl:The URL string used to execute the Content Script • node: the id of the Content Script object • useragent: the user's browser useragent • cookies: the user's browser cookies (as strings) • method: the HTTP verb used to request the script • lang: the user's locale • port: the HTTP port used to request the script • server: the HTTP host used to request the script • pathinfo: the request's URL path information
`request`	A synonym for the previous variable (for backward compatibility)

Support variables¶

The number and the nature of the variables that are injected by the Content Script Engine depends primarily from the mode through which the script has been executed. Content Script scripts used for example to implement Node Callbacks or columns' Data Sources will have injected in their Execution Context, respectively: the information regarding the Node that triggers the event or the Node for which the column's value is requested. Please refer to the Content Script module online documentation for the name and type of the variables made available in the Execution Context in the different scenarios. The following variables are always injected.

Variable	Description
`img`	Content Server static resource context path (es. /img/).
`webdav`	WebDav path
`supportpath`	Content server support path
`url`	Content Server CGI Context
`SCRIPT_NAME`	A synonym for the previous variable (for backward compatibility)
`csvars`	A map containing the script's static variables *(/working/contentscript/otcsobj/#static-variables)*
`originalUserId`	The ID of the user that triggered the execution of the Script (not considering impersonation)
`originalUsername`	The username of the user that triggered the execution of the Script (not considering impersonation)

IMG

Please note that most of the time the img context variable ends with a trailing slash. To correctly use it as a replacement variable in Content Script strings or velocity templates we suggest you to use the ${img} notation. E.g.:

```
"""<img src="${img}anscontentscript/img/am_logo.png" />"""
```

Support objects¶

Support objects are instances of Content Script classes that the Script Manager creates, configures and injects into every execution context in order to provide a simple mean for accessing very basic or commonly required functionalities.

Variable	Description
`self`	An object representing the Content Script node being currently executed.
`response`	An instance of the ScriptResponse class that can be used to pilot the Content Script output.
`gui`	A map of standard Content Server UI Components that can be enabled/disabled at the time of rendering the page. E.g. • gui.search = false • gui.sideBar = false

Disable standard UI

To completely disable the standard Content Server UI use:

Variable	Description
	 `gui.gui = false`
`log`	Each Content Script is associated with an instance logger that can be used to keep track of the script execution. From within a script you can access the logger either using the Script's method getLog() or the shortcut **log**. The Content Script logging system is based on a framework similar to the one used internally by OTCS. The logger supports five different levels: trace, debug, info, warn, error. The default log level for any script is: **error** this means that log messages at level for example **debug** won't be outputted in the ModuleSuite's master log file (cs.log). Logging level can be overridden per script basis through a dedicated administrative console.
`out`	A container for the script textual output

Base API¶

The Content Script "Base API" or "Script API" is constituted by methods and properties that are exposed directly by each Content Script script.

API	Description
`asCSNode(Map)`	An alternative to loading a node explicitly using one method out of: `docman.getNode`, `docman.getNodeByPath`, `docman.getNodeByNickname`
`asCSNode(Long)`	An alternative to loading a node explicitly using the `docman.getNode` method
`redirect(String)`	A shortcut for sending a redirect using the response object
`json(String)`	A shortcut for sending json using the response object
`json(Map)`	A shortcut for sending json using the response object
`json(List)`	A shortcut for sending json using the response object
`sendFile(File[,String])`	A shortcut for sending a file using the response object
`success(String)`	A shortcut for setting the result of the script execution to "success"
`runCS(Long)`	A utility method to run a second Content Script (identified by ID) within the same context
`runCS(String)`	A utility method to run a second Content Script (identified by nickname) within the same context

API	Description
`runCS(String, Object[])`	A utility method to run a second Content Script (identified by nickname) using a cleaned execution context (the new execution context shares with the caller's context only the Content Script services and the following variables: out, gui, response). In the sub-script code the parameters that have been used to call the sub-script can be accessed through the context variable "args". Using this variant it's possible to intercept the result of the sub-script execution.
`printError(Ex)`	A utility method to print out any exception raised by script's execution

Examples

Usage example for runCS(String, Object[]) API

```
//Parent Script
node = asCSNode(123456)
map  = runCS("mySubScript", node, users.current)
out << map.user

//SubScript "mySubScript"
def retVal = [:]
retVal.name = args[0].name
retVal.user = args[1].with{
 [
    name:it.displayName,
    id:it.ID
 ]
}
return retVal
```

Usage example for asCSNode(...)API

```
// Load a CSNode
asCSNode(2000)

// A node can be loaded also by path or nickname
asCSNode(nickname:"MyNode")
asCSNode(path:"path:to:myNode")
asCSNode(id:2000) //=== asCSNode(2000)
```

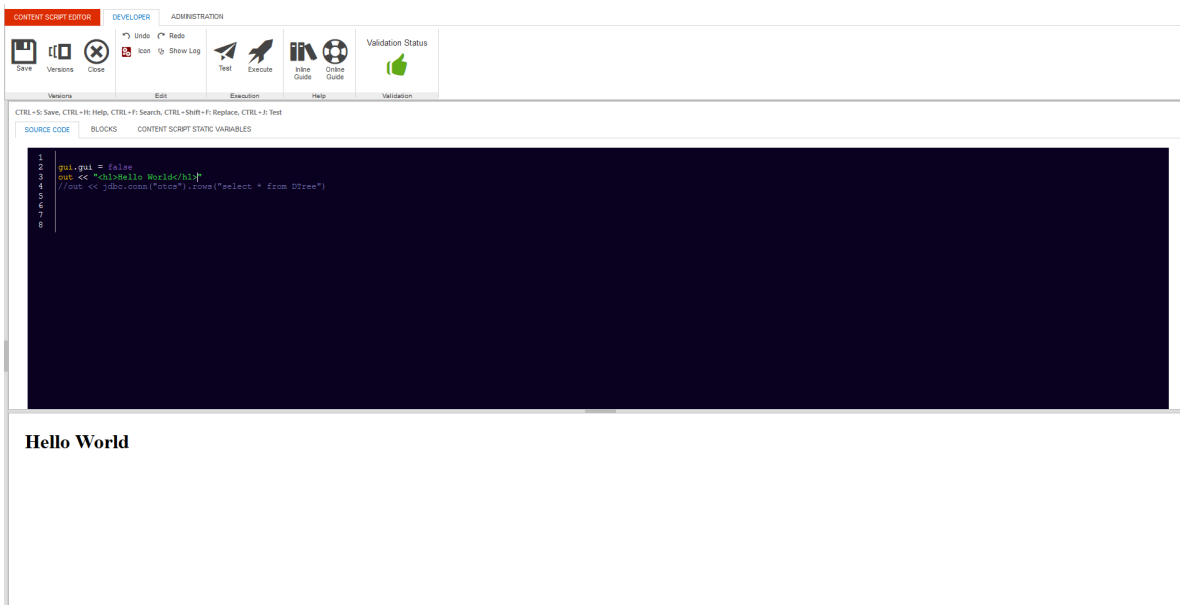
Usage example for printError(...)API

```
try{
    out << asCSNode(12345).name
}catch(e){
    log.error("Error ",e) //Prints the full stack trace in the log file
    printError(e) //Outputs the error

}
```

Script's execution¶

As shown in previous sections, the execution of Content Scripts can be triggered in different ways. Here after are a few examples:

- **Direct execution by a user**. This can happen, for example:
  - Using the **Execute** action in the object function menu or promoted object functions
  - While using the Content Script Editor, using the **Execute** or **Execute in Modal** buttons (useful for debug and testing purposes, shown in the figure below)
  - A **URL** associated to the execution of a Content Script is invoked
  - A **Content Script backed SmartUI widget** is displayed
- **Direct execution by an external system**
  - A URL associated to a **Content Script REST API** is invoked
- **Automatic execution by the system**. This happens when:
  - The script is **scheduled**, at the configured execution time
  - A **callback** is configured, and the associated event is triggered
  - A Content Script **Workflow step** is configured as part of a workflow, and the step is activated
  - A Content Script is configured as a **Data Source for a WebReport**, and the WebReport is executed
  - A Content Script serves as a **Data Source** for a custom column



Script's output¶

As you can easily imagine by analysing the examples in the previous paragraph, the expected result from the execution of a Content Script varies significantly from case to case.

When a user executes a Content Script directly from the Content Server user interface, he/she would probably expect, in most of the cases, the result to be either a **web page**, a **file to download**, or a **browser redirection** to a different Content Server resource.

When a remote system invokes a REST service API backed by a Content Script, it will most probably expect structured data in return (probably **XML** or **JSON data**).

When a Content Script is executed as part of a workflow and the next step is to be chosen depending on the execution outcome, the script will probably be expected to return a single variable of some kind (**a number or a string**) or an indication that the execution was either **successful** or encountered **errors**.

Content Script is flexible enough to cover all of these scenarios. The next section will include examples of how to provide the different output necessary in each situation.

HTML (default)¶

The default behaviour in case of a successful script execution is to return the content of the "out" container

```
def contentToPrint = "This content will be printed in output"
out << contentToPrint
```

```
def contentToPrint = "This content will be printed in output"

//If the object returned by the script is a String, it will be printed in output
return contentToPrint
```

JSON¶

JSON content can be easily returned

```
def builder = new JsonBuilder()
builder.companies {
    company "AnswerModules"
    country "Switzerland"
    }

// Stream JSON content, useful for restful services
response.json(builder)
```

```

```

```
String jsonString = '{"key":"value"}'

// A string containing JSON data can be used
response.json(jsonString)
```

```
// or with the shorthand method
json(jsonString)
// or
json([[key:”value1”], [key:”value2”]])
```

XML¶

XML content can be easily returned

```
gui.gui = false
gui.contentType = "application/xml"

def builder = new StreamingMarkupBuilder()
def parent  = asCSNode(2000)
def nodes   = parent.childrenFast //nodes are lazy loaded

def xml = builder.bind {
    node(id:parent.ID, name:parent.name,  isContainer:parent.isContainer){
        children {
            nodes.collect {
                node(id:it.ID, name:it.name, isContainer:it.isContainer)
            }
        }
    }
}
out << XmlUtil.serialize(xml)
```

Output of the above script:

```
<node id="2000" name="Enterprise" isContainer="true">
    <children>
            <node id="90064"    name="Import"   isContainer="true"/>
            <node id="3270165"  name="Training" isContainer="true"/>
    </children>
</node>
```

Using gui support object for tuning script's output

Note the usage of `gui.contentType` in order to change the response's “Content-Type” header.

Files¶

It is also possible to stream a file directly:

```
// Stream a file as result of the execution
def res  = docman.getTempResource("tempRes", "txt")
res.content.text = "Just a test"
```

```
def file = res.content
response.file(file)
```

```
// Stream a file as result of the execution
def res  = docman.getTempResource("tempRes", "txt")
res.content.text = "Just a test"
def file = res.content
// Stream a file, specifying if it is a temporary file (will prevent deletion)
response.file(file, true)
```

```
// Stream a file as result of the execution
def res  = docman.getTempResource("tempRes", "txt")
res.content.text = "Just a test"
def file = res.content
// or with the shortcut method
sendFile(file)
```

```
// Stream a file as result of the execution
def res  = docman.getTempResource("tempRes", "txt")
res.content.text = "Just a test"

// or returing the CSResource directly
res.name = "My textFile.txt"
return res
```

Managed resources¶

In the context of developing against OTCS you will end up dealing with many different kind of contents most of which are (or are strictly related with) files. In order to reduce the amount of code needed to properly manage the disposition of temporary files, Content Script introduces the concept of "managed resource" or CSResource. A CSResource is basically a wrapper around the File class. CSResources are managed by the Content Script engine (no disposition required) and are returned any time you want to access the content of a CSDocument or you fetch a version from it (in these cases the CSResource will keep a reference, towards the source CSDocument, through its "owner" property.

CSResources are first class citizens in Content Script. A CSResrouce can be for example returned directly by a Content Script, triggering the download of the same.

Returning CSResource to trigger document download

Returning a CSResource from a script is the simplest way to stream out a file in this case is important to keep in mind that the name of the downloaded file will be determined using the following rule:

```
if the property onwer of the CSResource is != null
then
    use the name of the CSNode referenced by the CSResource's owner property
else
    use the CSResource's name property.
end
```

Redirection¶

In alternative, the response could contain a redirection to an arbitrary URL:

```
String url = "http://www.answermodules.com"

// Send a redirect using the response
response.redirect(url)


// or with the shortcut method
redirect(url)
// or
redirect “${url}/open/2000”
// or
redirect asCSNode(2000).menu.open.url
```

HTTP Code¶

In certain cases (e.g. when Content Script is used to extend OTCS' REST APIs), it could be necessary to explicitly control the "error" or "success" status of the script execution:

```
// Force the script execution result to be "success" using the response
response.success("This is a success message")
response.success("This is a success message",200)

// or with the shortcut method
success("This is a success message")
success("This is a success message",200)

// Force the script execution result to be "success"
response.error("This is an error message", 403)

// or with the shortcut method
error("This is an error message", 403)
```

Advanced programming¶

Templating¶

Content Script features a flexible yet powerful templating engine based on Apache Velocity. Evaluating a template is just a matter of invoking one of the evaluate methods available through the template service.

Content Script velocity macros¶

Content Scripts defines a collection of macros that simplify the creation of OTCS UI embeddable interfaces. A developer can create his own macros simply defining them in a z_custom.vm file to be stored under the Content Script "Temp" folder (as defined in the Base Configuration page: amcs.core.tempFilePath).

Name and description	Param	Type and description	Usage example
csmenu(dataid[,nextUrl]) Creates the standard OTCS context menu for the given node (identified by its dataid)	dataid	Integer node's dataid	`#csmenu(2000)`
	nextUrl	String	
csresource(retList) Loads static libraries from the module support directory	resList	List A list of resources to load. To be chosen from: query, jquery-ui, jquery-ui-css, bootstrap, bootstrap-css	`#csresource([‘bootstrap’])`

csform(script[,submit])
Creates the HTML form needed to submit a request against the executed Content Script
	script	Integer
		The objId of the Content Script you’d like to execute
			`#@csform()`
			`//Custom form inputs go here`
			`#end`

	submit	String The value for the label of the submit button. If null the submit button will not be created	

cstable(columns,sortColumn, columnsClasses[,checkBoxes])
Creates an HTML table that fits nicely with the standard OTCS UI
	columns	List
		The list of column labels
			`#@cstable([‘First Name’], {},{}, true)`
			`//Your rows here`
			`#end`

	sortColumns	Map A map of “Column Label”, “Property” couples. The Property is used to build	

Name and description	Param	Type and description	Usage example
		sort links for columns	
	columnsClasses	Map A map of “Column Label”, “CSS Classes” couples. The “CSS Classes” are assigned to the THs tags.	
	checkBoxes	Boolean If TRUE the first column of the table will have width 1%. To be used to insert a checkboxes column	
cspager(skip,pageSize, pagerSize,elementsCount) Creates a pagination widget to be used	skip	Integer The index of the element to skip before to start rendering rows	`#cspager(0 25 3 $parent.childCount)`
	pageSize	Integer The page size (e.g. 25)	
	pagerSize	Integer The number of pages to show in the pager widget	
	elementsCount	Integer The total	

Name and description	Param	Type and description	Usage example
		number of elements	

OScript serialized data structures¶

Content Script Java layer is tightly bound with Content Script Oscript layer, thus quite frequently you will face the need of managing Oscript's serialized data structures obtained for example querying the OTCS' database or from nodes' properties.

Oscript serializes its data in the form of Strings, for this reason Content Script enhances the String class in order to provide a quick method for retrieving the corresponding Content Script's objects out of the OScript serialized representation.

Methods available on the String class are:

- getDateFromOscript
- getListFromOscript
- getMapFromOscript

In the exact same way Content Script enhances its most common types (List, Map, Date, Long, CSReportResult) in order to simplify the creation of the corresponding OScript serialized representation.

The below table shows an usage example of the mentioned features:

Statement	Result
"D/2011/7/19:18:10:51".getDateFromOscript()	Tue Jul 19 18:10:51 CEST 2011
"{1,2,3}".getListFromOscript()	[1, 2, 3]
"A<1,?,'key1'=1000,'key2'={1,2,A<1,?,'key3'=2002,'key4'=D/2017/7/19:18:10:51>}>".getMapFromOscript()	[key2:[1, 2, [key4:Wed Jul 19 18:10:51 CEST 2017, key3:2002]], key1:1000]
sql.runSQLFast("select ExtendedData EXT from DTree where DataId = %1",false,false, -1, 520305).rows[0].EXT.getMapFromOscript()	[DisplayAsLink:false, alignment:right, dataSource:attr_93202_3, NewWindow:false, sortable:false, DisplayValue:%value%, inheritedPermID:93202, columnDisplayWidth:20, locations:[90372], TitleText:, indexName:null, longText:0, columnEMWidth:9, columnName:WND_attr_93202_3, URL:?func=ll&objId=%objid%&objAction=attrvaluesedit&version=-1&nexturl=%nexturl%]
sql.runSQLFast("select DATAID, NAME from DTree where DataId = %1",false,false, -1, 520305).getOscriptSerialization()	V{<'DATAID','NAME'><520305,'Amount'>}
"January 1, 2013".asDate().getOscriptSerialization()	D/2013/1/1:12:0:0
[1,2,3].getOscriptSerialization()	{1,2,3}
['key1':1000, 'key2':[1,2, ['key3':2002, 'key4':new Date()]]].getOscriptSerialization()	A<1,?,'key1'=1000,'key2'={1,2,A<1,?,'key3'=2002,'key4'=D/2017/7/20:9:52:43>}>

Optimizing your scripts¶

Behaviors¶

You can use behaviors to decorate your scripts and let them implement a specific set of new functionalities. Behaviors are to be considered similar to inheritance. A behavior is defined as a

collection (MAP) of closures and usually implemented in the form of a static class featuring a **getBehaviors** method.

When you add a behavior to your script, all the closures that have been defined in the behavior become part of your script thus becoming part of your script context.

Behaviors are resolved at compilation time, this means that they should be considered as a static import.

Said otherwise, any changes applied directly on the script that implements your behaviors, won't effect the scripts that have imported such behaviors. In order to update the imported behaviors you have to trigger the re-compilation of the script that is importing them (target script).

BehaviorHelper¶

In order to add behaviors to a script you shall use the BehaviourHelper utility class.

The BehaviourHelper utility class, features three methods:

```
@ContentScriptAPIMethod (params = [ "script" , "behaviours" ], description = "Add behaviours to a Co
public static void addBeahaviours(ContentScript script, Map<String, Closure> closures)

@ContentScriptAPIMethod (params = [ "script" ,  "behaviours" ], description= "Remove behaviours from
public static void  removeBehaviours(ContentScript script, String... closures=null)

@ContentScriptAPIMethod (params = [ "script" ,  " behaviour " ], description= " Determine if the scr
public static void  hasBehaviour(ContentScript script, String name)
```

Through BehaviourHelper you can add, remove or check for the presence of an associated behavior.

Behaviors are of great help when it comes to structure your code base, optimize executions and reduce boilerplate code.

Module Suite comes with few predefined behaviors, you can easily implement yours by defining a map of closures to be passed to the above BehaviourHelper utility class.

Default Behaviours¶

The AMController behavior has been designed to simplify the creation of form-based application on Content Server.

It features the following closures:

1. **start**: this closure takes no parameters, and it is used to dispatch incoming requests. It creates (if not already provided) an app object to be made available in the execution context. It analyzes the request's pathinfo, to extract the information required to route towards a registered closure. Rebuilds any Beautiful WebForm object found in the request.

This closure should be the last instruction of your script.

```
app =  [:]
app.product ="Module Suite"

if(!BehaviourHelper.hasBehaviour(this, "start") ) {
    BehaviourHelper.addBeahaviours(this, AMController.getBehaviours())
}

home = {
    out << "Hello world from ${app.product}"
}

details = { String id = null->
    out << "This script ID ${id?asCSNode(id as int).ID:self.ID}"
}
start()
```

When directly executed (`http://my.server/otcs/cs.exe?func=ll&objId=12345&objAction=Execute&nexturl=..` or `http://my.server/otcs/cs.exe/open/12345`) the script above will output:

```
Hello world from Module Suite
```

when executed using: `http://my.server/otcs/cs.exe/open/12345/details` it will output

```
The script ID 12345
```

when executed as: `http://my.server/otcs/cs.exe/open/12345/details/2000` it wll output:

```
The script ID 2000
```

In other words the requested path will always been interpreted using the follow schema: `http://my.server/otcs/cs.exe/open/12345/closurename/param1/param2/param3` where **closurename** will be defaulted to "home" if not found in the path.

2. **loadForm(def formID, def amSeq=0):** loads a Form data object, setting form.viewParams.contentScript = params.node (so that if the form data object will be used with a BeautifulWebForm view the form will submit on this very same content script) and form.viewParams.amapp_Action = params.pathinfo.

3. **submitForm(def form):** validates the form data object and performs the submit (executing pre-submit and on-submit scripts if defined)

4. **renderForm(def form, def context=null):** renders the form either in the script context or in the specified context

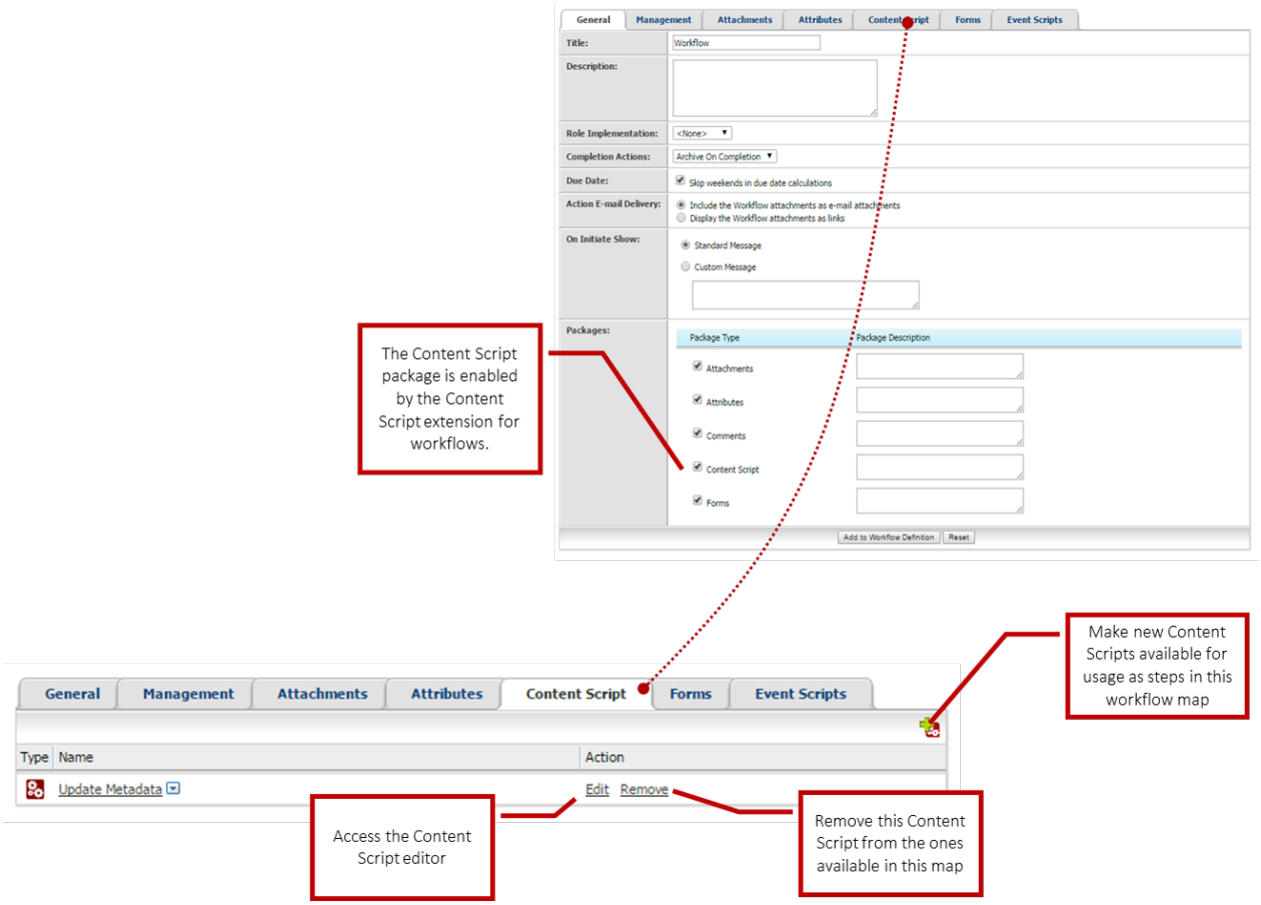
Working with workflows

Content Script Workflow Steps¶

The Content Script Extension for Workflows is automatically available upon installation of the Content Script module. The extension enables a new workflow package in Workflow maps (Content Script package) and custom type of workflow step (Content Script step).

Content Script Package¶

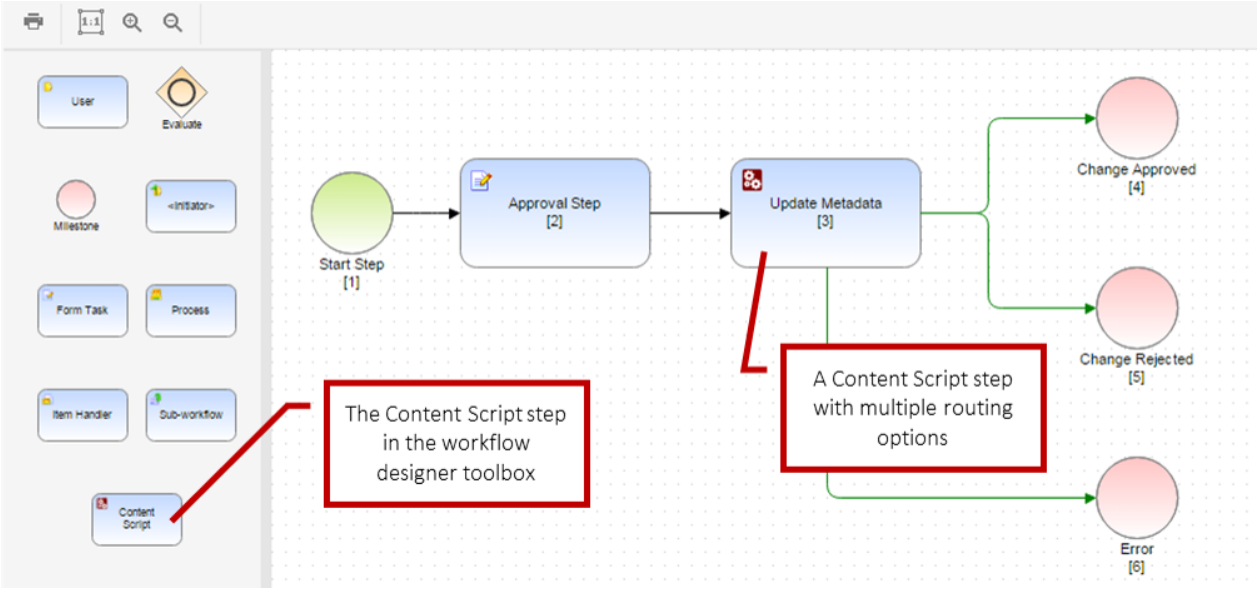
The Content Script package must be enabled in order to use Content Script steps within a workflow map.



Once enabled, it will be possible to define the set of Content Script objects that will be available for inclusion in the current workflow map.

Content Script Workflow Step¶

Content Scripts enabled in the workflow package can be used in the workflow map as Content Script steps.



Here below is an example of a Content Script step performing some basic operations on the current workflow task.

```
// Fetch the menu in its original format
def workflowStatus  = workflow.getWorkflowStatus(workID, subWorkID)
def workflowTask  = workflow.getWorkFlowTask(workID, subWorkID, taskID)
def allTasks   = workflowStatus.tasks

// Edit Workflow Attribute values
def workflowAttributes = workflowStatus.getAttributes()
workflowAttributes.setAttributeValues("Customer", "ACME inc.")
workflowAttributes.setAttributeValues("Country", "Switzerland")
workflow.updateWorkflowData(workID,  subWorkID, [workflowAttributes]) //Updates attributes

// Edit  Workflow Attribute values - different flavour
try{

    def atts =workflowStatus.getAttributes()

    // This API is not just for reading values...
    // Set the value
    atts.data.Customer = "ACME inc."
    atts.data.Country = "Switzerland"

    workflowStatus.updateData() // COMMIT CHANGES

}catch(e){
    log.error("Unable to access workflow's attributes ",e)
}

// Access a workflow form
def form = forms.getWorkFlowForm(workflowTask, "Form")
form.myattribute.value = "A new value"
forms.updateWorkFlowForm(workflowTask, "Form", form, false)

// Update Task's title
workflow.updateTaskTitle(
                          workID,
                          subWorkID,
                          taskID,
                          "Title with form field: ${form.myattribute.value}"
                        )

// Access a workflow form and workflow attributes - different flavour
```

```
//Mapping
node = asCSNode(path:"Some Path:On Content Server:Node")

workflowStatus.attributes."Account Folder" = node.ID
workflowStatus.forms.Form.data."Lead Owner" = node.Account."Account Manager"
workflowStatus.forms.Form.data."Company" = node.Account."Company name"
workflowStatus.forms.Form.data."First Name" = node.Account."Contacts"."First Name"
workflowStatus.forms.Form.data."Last Name" = node.Account."Contacts"."Last Name"
workflowStatus.forms.Form.data."Email" = node.Account."Contacts"."Email"
workflowStatus.forms.Form.data."Addresses"."Street" = node.Account."Addresses"."Street"
workflowStatus.forms.Form.data."Addresses"."City" = node.Account."Addresses"."City"
workflowStatus.forms.Form.data."Addresses"."Zip Code" = node.Account."Addresses"."ZipCode"
workflowStatus.forms.Form.data."Addresses"."Country" = node.Account."Addresses"."Country"
workflowStatus.updateData()   // COMMIT CHANGES

// Updating Workflow title
workflow.updateWorkFlowTitle(
                              workID,
                              subWorkID,
                              "Company: ${workflowStatus.forms.Form.data."Company" as String}"
                             )

// Add documents to the attachments folder (an empty spreadsheet in this case)
def workflowAttachments = workflowStatus.getAttachmentsFolder()
workflowAttachments.createDocument("Spreadsheet", xlsx.createSpreadsheet().save())
```

In the above example, the script is:

- fetching information related to the current workflows status and tasks
- performing changes on some workflow attributes
- fetching and updating a workflow form
- adding attachments to the workflow attachments folder

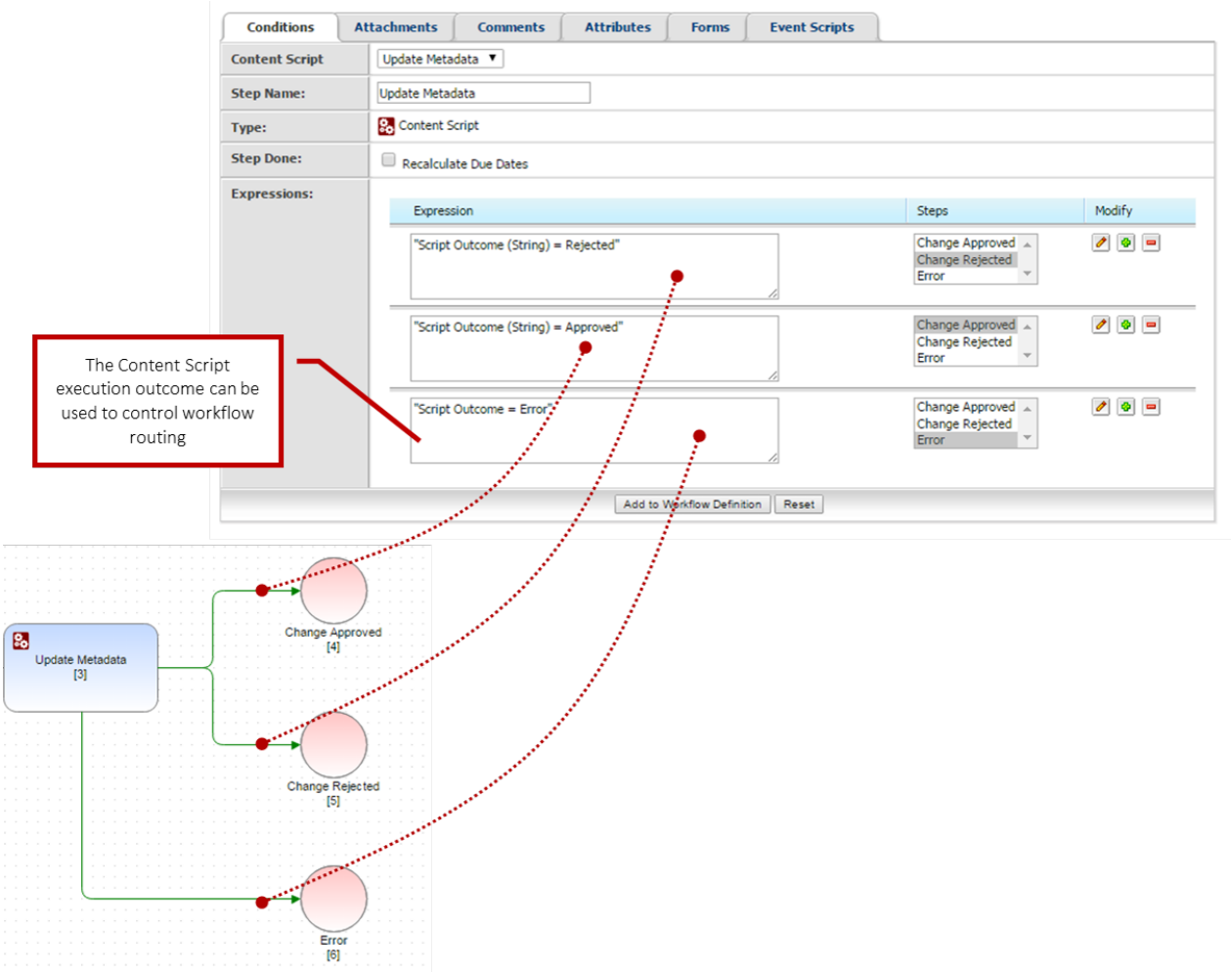
Note that the above script makes use of some context variable available in the execution context that are peculiar only to workflow steps. The variables are:

Expression type	Type	Description
workID	Integer	The workflow ID
subWorkID	Integer	The subworkflow ID
taskID	Integer	The current task ID

The above variables can be used in combination with the **workflow** service API to access all the information related to the current workflow. See the complete API documentation for a complete list of operations available on workflow instances.

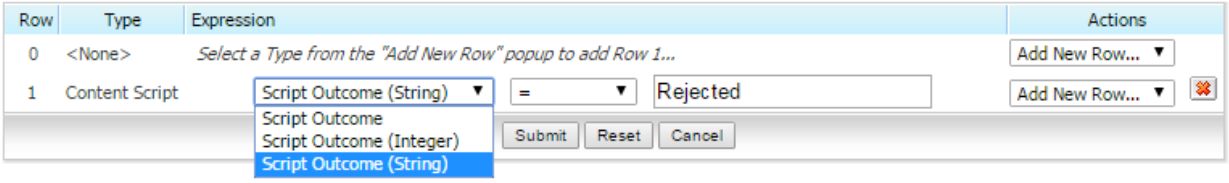
Workflow routing¶

Content Script execution outcome, which MUST always be a String, can be interpreted in different ways, and used to route the next steps of the workflow.



The following routing expression types are currently supported:

Expression type	Values	Description
Content Script Outcome	Success or Error	Error in case the script returns an exception
Content Script Outcome (Integer)	Any Integer value	Supports evaluation based on numeric comparison
Content Script Outcome (String)	Any String value	Evaluation based on string comparison



Synchronous and Asynchronous callbacks¶

Since version 1.5, Content Script supports the definition of **Event Callbacks**: in response to specific actions performed on Content Server, it is possible to execute one or more Content Scripts.

The callbacks can be:

- **synchronous**: the script is executed within the same transaction as the triggering action. Synchronous callbacks are configured through the **CSSynchEvents** container.
- **asynchronous**: the triggering action completes normally. The callback script is executed later on. Asynchronous callbacks are configured in the **CSEvents** container.

Since synchronous callbacks are performed in the same transaction as the event, any errors that occur during script execution will cause the transaction to roll back.

Performance

Since synchronous callbacks are executed in the same transaction as the event, make sure that any action performed by the script requires a reasonable time span for execution. Otherwise, the user experience could be affected negatively.

Synchronous Callbacks are disabled by default

Please read the instruction below about how to enable them.

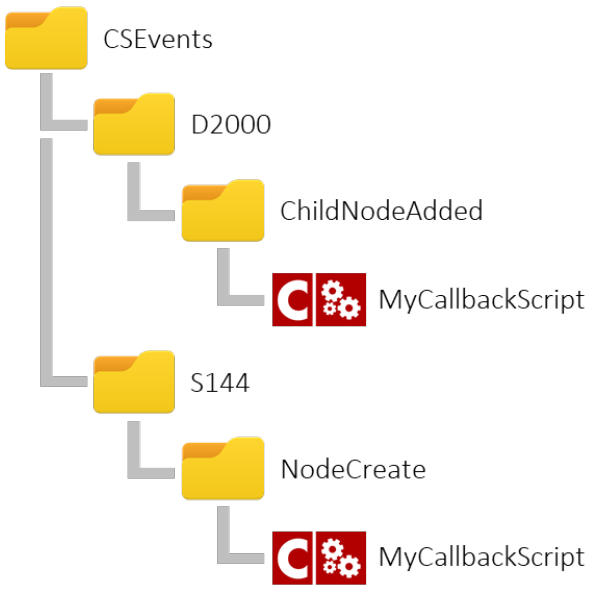
The definition of Content Script callbacks is based on a convention over configuration approach. In order to register a new callback, a script should be placed somewhere in a nested container structure in the **CSSynchEvents** or **CSEvents** container, following a specific naming convention.

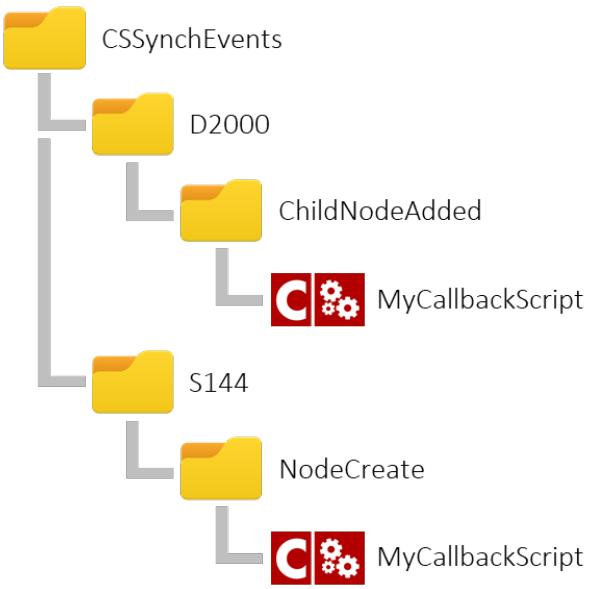
The first level under the container indicates the object or object subtype to which the callbacks are bound.

The naming convention is one of the following:

- `D<nodeID>`
- `S<subtype>`

where **nodeID** identifies the node unequivocally and **subtype** identifies a specific object subtype on Content Server.





Examples:

D2000 will intercept events on the Enterprise Workspace

S144 will intercept event on Document type objects (subtype: 144)

The second level should be once again a container and specifies the **event type**. The name of this container should be one of:

- ChildNodeAdded
- ChildNodeCreate
- NodeAddVersion
- NodeAddVersionPre
- NodeCopy
- NodeCreate
- NodeCreatePre
- NodeMove
- NodeRename
- NodeUpdate
- NodeUpdateCategories

Inside the Event Type container it is possible to place one or more Content Scripts that will be invoked when the callback is triggered.

The Module Suite Administration pages feature a Manage Callbacks tool that can be used to verify, at any time, all the callbacks that are bound to a specific object or subtype.

In the following tables we present a summary of the supported Events and the information regarding the variables that are injected in the Execution Context, automatically by the framework, for each event. These variables can be useful to implement the required business logic within the Script.

Event Name	Execution Context Param	Type	Description
All	callbackID	String	The CSEvent Name (NodeAddVersion, NodeUpdateCategories, etc)
	eventSourceID	Integer	The dataid of the node that triggered the event
NodeAddVersion	nodeID	Integer	The document that has received the new version
NodeAddVersionPre	nodeID	Integer	The document that has received the new version
NodeUpdateCategoriesPre	nodeID	Integer	The updated node's id
	addedCategories	List	The list of added categories
	deletedCategories	List	The list of removed categories
	changes	ChangeAssoc	The list of applied attributes changes. (SEE THE TABLE BELOW FOR DETAILS)
NodeUpdateCategories	nodeID	Integer	The updated node's id
	addedCategories	List	The list of added categories
	deletedCategories	List	The list of removed categories
	changes	ChangeAssoc	The list of applied attributes changes.

Event Name	Execution Context Param	Type	Description
			(SEE THE TABLE BELOW FOR DETAILS)
NodeCopy	nodeID	Integer	The id of the node that has been copied
	newNodeID	Integer	The newly created node's id
ChildNodeAdded	nodeID	Integer	The id of the node where a new content has been added
	newNodeID	Integer	The newly created node's id
NodeCreatePre	newNodeID	Integer	The newly created node's id
	nodeID	Integer	The newly created node's id
	addedCategories	List	The list of added categories
	deletedCategories	List	The list of removed categories
	changes	ChangeAssoc	The list of applied attributes changes. (SEE THE TABLE BELOW FOR DETAILS)
NodeCreate	newNodeID	Integer	The newly created node's id
	nodeID	Integer	The newly created node's id
	addedCategories	List	The list of added categories
	deletedCategories	List	The list of removed categories
	changes	ChangeAssoc	The list of applied attributes changes. (SEE THE TABLE BELOW FOR DETAILS)

Event Name	Execution Context Param	Type	Description
ChildNodeCreatePre	nodeID	Integer	The id of the node where a new content has been added
	newNodeID	Integer	The newly created node's id
	addedCategories	List	The list of added categories
	deletedCategories	List	The list of removed categories
ChildNodeCreate	nodeID	Integer	The id of the node where a new content has been added
	newNodeID	Integer	The newly created node's id
	addedCategories	List	The list of added categories
	deletedCategories	List	The list of removed categories
	changes	ChangeAssoc	The list of applied attributes changes. (SEE THE TABLE BELOW FOR DETAILS)
NodeMove	nodeID	Integer	The moved node's id
NodeRename	nodeID	Integer	The renamed node's id
	oldName (**Can be null**)	String	The previous node's name
	newName	String	The current node's name
NodeUpdate	nodeID	Integer	The updated node's id
NodeRenditionNew	nodeID	Integer	The node's id that received the new rendition
BusinessWorkspaceCreate	newNodeID	Integer	The newly created Business Workspace's id

Event Name	Execution Context Param	Type	Description
	nodeID	Integer	The newly created node's id
BusinessWorkspaceUpdate	nodeID	Integer	This callback is called when the business workspace update is about to complete.
BusinessWorkspaceChangeReference	nodeID	Integer	This callback is called in the context of workspace reference being updated (as in add or change).
BusinessWorkspaceRemoveReference	nodeID	Integer	This callback is called in the context of workspace reference being removed.
BusinessWorkspaceRelationsUpdate	nodeID	Integer	This callback is called in the context of workspace reference being updated (as in add or change).
	updateInfo	Map	A map with keys: childrenAdded, childrenRemoved, parentsAdded, parentsRemoved

The following table is related to the structure of the **ChangeAssoc** object, necessary to manage **NodeUpdateCategories** type events.

Property name	Type	Description
attributePath	List	The path of the modified attribute inside the category.
{"Name",0}: represents the path to the first value of the attribute "Name"		
{"Name",1}: represents the path to the second value of the attribute "Name"		
{"Addresses", 2, "ZipCode", 0}: represents the path to the first value of the attribute ZipCode in the third		

Property name	Type	Description
occurrence of the Set attribute Addresses		
oldValue	Dynamic	The previous attribute's value
newValue	Dynamic	The present attribute's value
categoryName	String	The category name

Synchronous Callbacks Configuration¶

Default Settings¶

Synchronous callbacks can significantly impact system performance. Therefore, they are **disabled by default** to ensure system stability.

Enabling Synchronous Callbacks¶

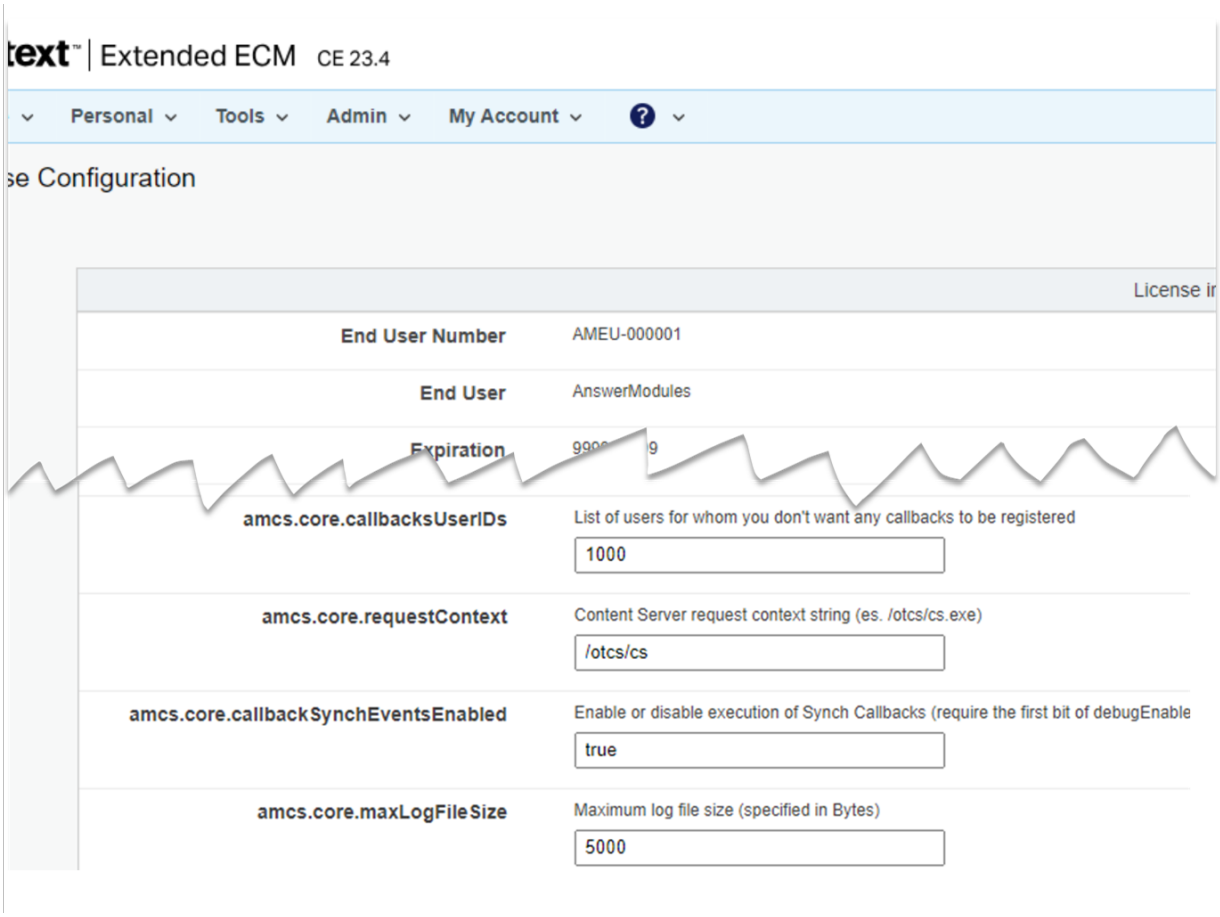
To enable synchronous callbacks, set the following property: `amcs.core.callbackSynchEventsEnabled = true`

User-Specific Configuration¶

In certain scenarios, it's beneficial to exclude specific users from synchronous callbacks, especially those performing bulk jobs. This exclusion helps in maintaining system efficiency and avoiding unnecessary load.

Specifying Excluded Users¶

- To exclude users, add their IDs to the `amcs.core.callbacksUserIDs` property.
- Multiple user IDs can be specified, separated by commas.
- Example format: `12345,6789`



InterruptCallbackException - transaction roll-backed¶

There are cases in which you might want your synchronous callback to cause the roll-back of the original event transaction (to prevent its completion), e.g. you implemented a synch-callback triggered by the NodeCreate event and you want to use it to ensure that the node that is going to be created respects some specific business rule, for example, it's a PDF document. In this cases, you can just raise an un-catched InterruptCallback exception from within your callback script.

E.g.

```
log.error("Running ${self.parent.parent.name}:${self.parent.name}:${self.name} for $nodeID")
out << "This is the mother of all failures..."
throw new InterruptCallbackException("New Callback Exception...")
```

Returning meaningful messages to your users

To return a message to your users you have just to add an output statement to your script.

Extending REST APIs

Extending REST APIs:CSServices¶

The **CSServices** container is dedicated to Content Scripts that should be made available as REST services.

The name of scripts placed in this container can be used to invoke the script directly through two dedicated HTTP endpoint (**amcsapi**, **amcsapi/v1**)

The **amcsapi** can be used to consume the REST service from within the Content Server GUI (it will in fact use the standard Content Server authentication mechanism to authenticate the user).

On the other hand the **amcsapi/v1** can be used to consume the REST service using the Content Server REST Apis authentication token *(https://developer.opentext.com/webaccess/#url=%2Fawd%2Fresources%2Farticles%2F6102%2Fcontent%2Bserver%2Brest%2Bapi%2B%2Bquick%2Bstart%2B*

> When invoked, unless otherwise specified (for example, in the script's "Run As" configuration), each script is executed as the currently logged in user.

Basic REST service¶

As a very simple example, the script **getuserbyname** can be invoked by using an URL built as follows:

```
http://localhost/otcs/cs.exe/amcsapi/getuserbyname
```

```
http://localhost/otcs/cs.exe/amcsapi/v1/getuserbyname
```

Additional parameters can be passed to the service, and will be available in the Content Script (via the **params** object). For example, invoking the previous script as:

```
http://localhost/otcs/cs.exe/amcsapi/getuserbyname?term=admin
```

the REST service framework will run the backing **getuserbyname** script adding the value of the GET parameter **term** in the **params** container variable. In the script, the value will be accessible by simply using the expression:

```
params.term
```

Behaviour based REST services¶

Since version 1.7.0, Content Script supports a “behaviour” based approach for the creation of REST services. This allows for an easier set-up of new services, enhance maintainability and better compliance with REST service commonly used conventions and de-facto standards.

A skeleton for a behaviour-based REST service is shown below.

A REST service can specify multiple operations, identified with behaviours. Each behaviour is implemented as a *closure*. By convention, the **home** behaviour is bound to the root of the API.

Service example¶

```
log.debug("Content Script REST Service {} - START", self?.name)

section = { String elemID=null, String method=null, String param=null ->
    try {
        if(elemID){
            switch(params.method){
                case "GET": //Read
                json(
                    [
                        operation:"section",
                        elemID:elemID,
                        method: method,
                        param: param
                    ]
                )
                return;
                default :
                response.error("Unsupported operation",500)
                return
                    }
        }else{
            json(
                [
                    operation:"section",
                    elemID:elemID,
                    method: method,
                    param: param
                ]
            )
            return
                }
    } catch(e){
        log.error("An error has occurred while managing the request", e)
        json([error:"Unable to complete your request $e?.message"])
    }
}

//Default service method
home = { String elemID ->
    try {
        //Single element
        if(elemID){
            switch(params.method){  //request verb
                // CRUD operations
                case "POST": //Create
                //Your code here...
                break;
                case "GET": //Read
                json(["elemID":elemID])
```

```
                return;
                case "PUT": //Update
                //Your code here...
                break;
                case "DELETE": //Delete
                //Your code here...
                break;
            }
        }else{
            switch(params.method){  //request verb
                // CRUD operations
                case "POST": //Create
                //Your code here...
                break;
                case "GET": //Read
                //Your code here...
                break;
                case "PUT": //Update
                //Your code here...
                break;
                case "DELETE": //Delete
                //Your code here...
                break;
            }
        }
        // Default return
        json([ok:true])
    } catch(e){
        log.error("An error has occurred while managing the request", e)
        json([error:"Unable to complete your request $e?.message"])
    }
}

if(!BehaviourHelper.hasBehaviour(this, "start")) {
    BehaviourHelper.addBeahaviours(this, AMRestController.getBehaviours())
}

return start()

log.debug("Content Script REST Service {} - END", self?.name)
```

Sample invocation path	Operation	Parameters passed to the closure
/training	home	elemID = null
/training/2000	home	elemID = “2000”
/training/2000/section	section	elemID = “2000” method = null param =null
/training/2000/section/100	section	elemID = “2000” method = “100” param =null
/training/2000/section/100/list	section	elemID = “2000” method = “100” param =“list”
/training/section/2000	section	

Sample invocation path	Operation	Parameters passed to the closure
		elemID = “2000” method = null param =null
/training/section/2000/100	section	elemID = “2000” method = “100” param =null
/training/section/2000/100/list	section	elemID = “2000” method = “100” param =“list”

Extending Content Script

Create a Custom Service¶

One of the most important feature of ModuleSuite is its extensibility. ModuleSuite has been in fact designed in order to let you extend it, creating new services, new components, widgets, code snippets etc..

Creating a new service it's particularly helpful when it comes to integrate other services and/or systems, or to leverage existing libraries to extend the Content Server capabilities. Creating your extension in the form of a new Content Script service you will automatically benefit from all the existing ModuleSuite features such as, for example, the full support of the Content Script Editor.

New services can be easily created by using the Content Script SDK. The **Content Script SDK** is a toolkit that can be used by developers to **create custom Content Script services**. Services created with the SDK can be seamlessly deployed in the target Content Server instance, and be accessible within Content Script code.

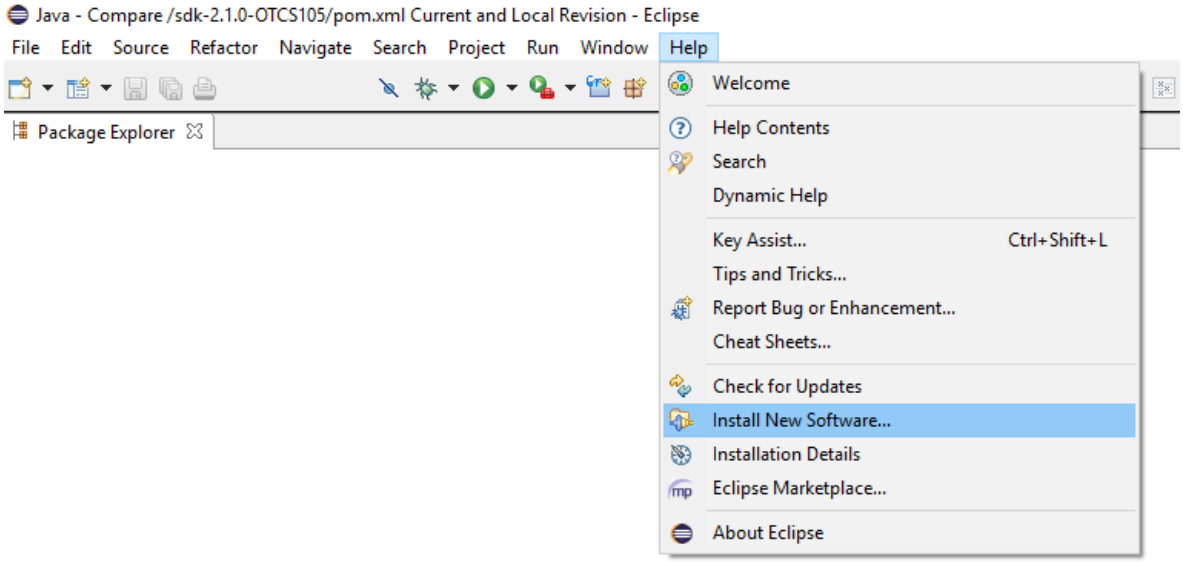
The suggested way to setup and use the Content Script SDK is by using the well-known **Eclipse IDE**.

The SDK is shipped in the form of an Eclipse Maven project. The project includes all the interfaces required for integration within Content Script, and can be used as a template to create a custom service.

Content Script SDK setup¶

1. Download **Eclipse Luna SR2** (https://eclipse.org/downloads/packages/eclipse-ide-java-developers/lunasr2c *(https://eclipse.org/downloads/packages/eclipse-ide-java-developers/lunasr2c)*) *

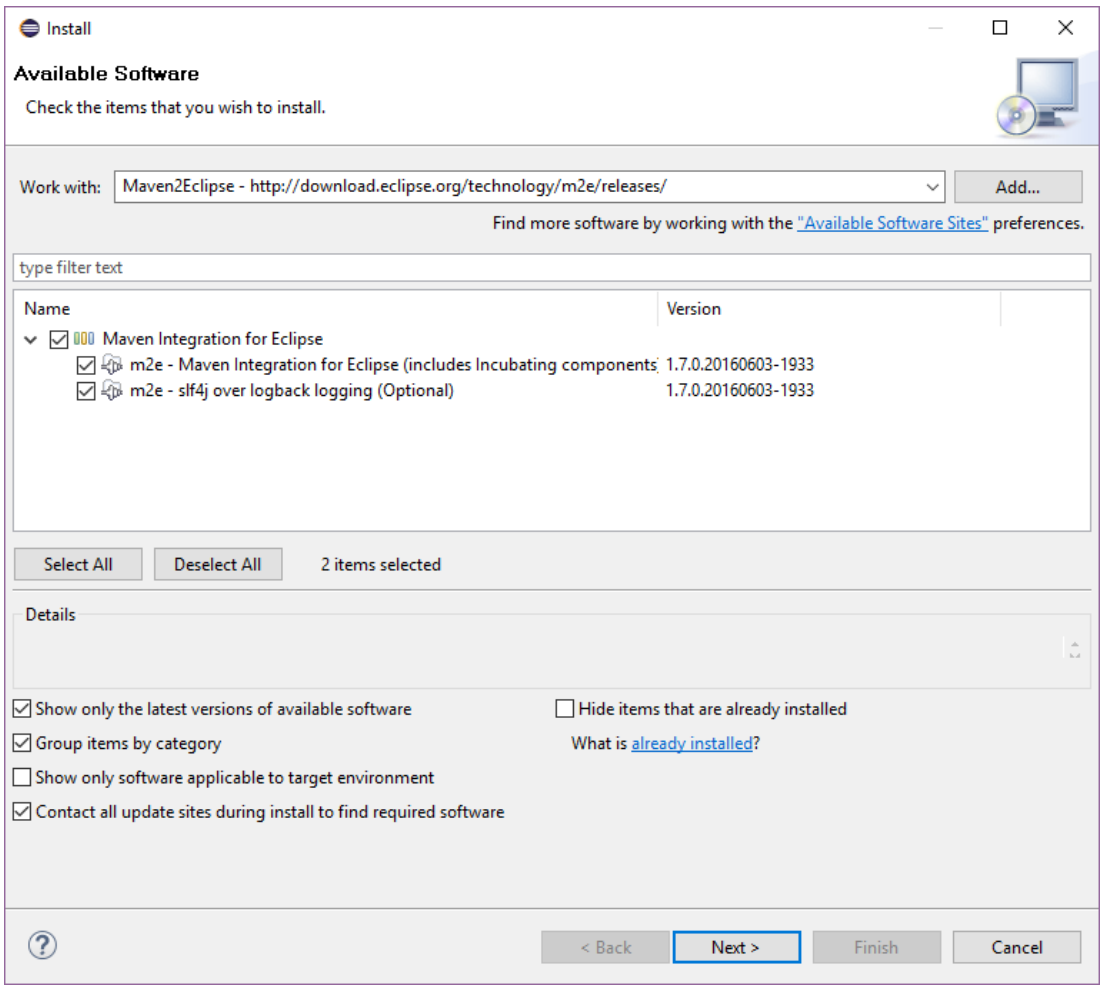
2. Run Eclipse. Use the **Help > Install new software** option to install some required additional components



3. Install **Maven2Eclipse** components

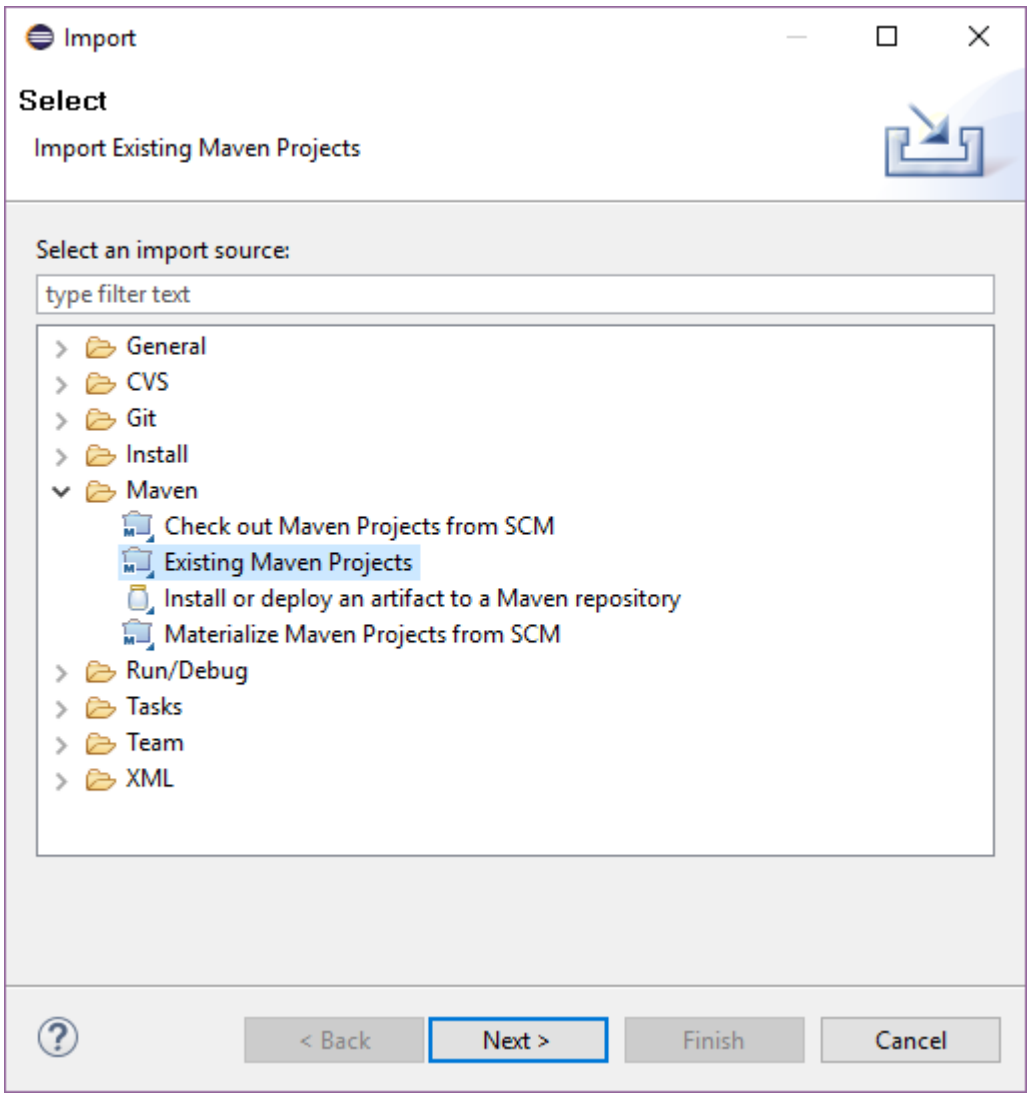
    1. add the update site (http://download.eclipse.org/technology/m2e/releases/ *(http://download.eclipse.org/technology/m2e/releases/)*)

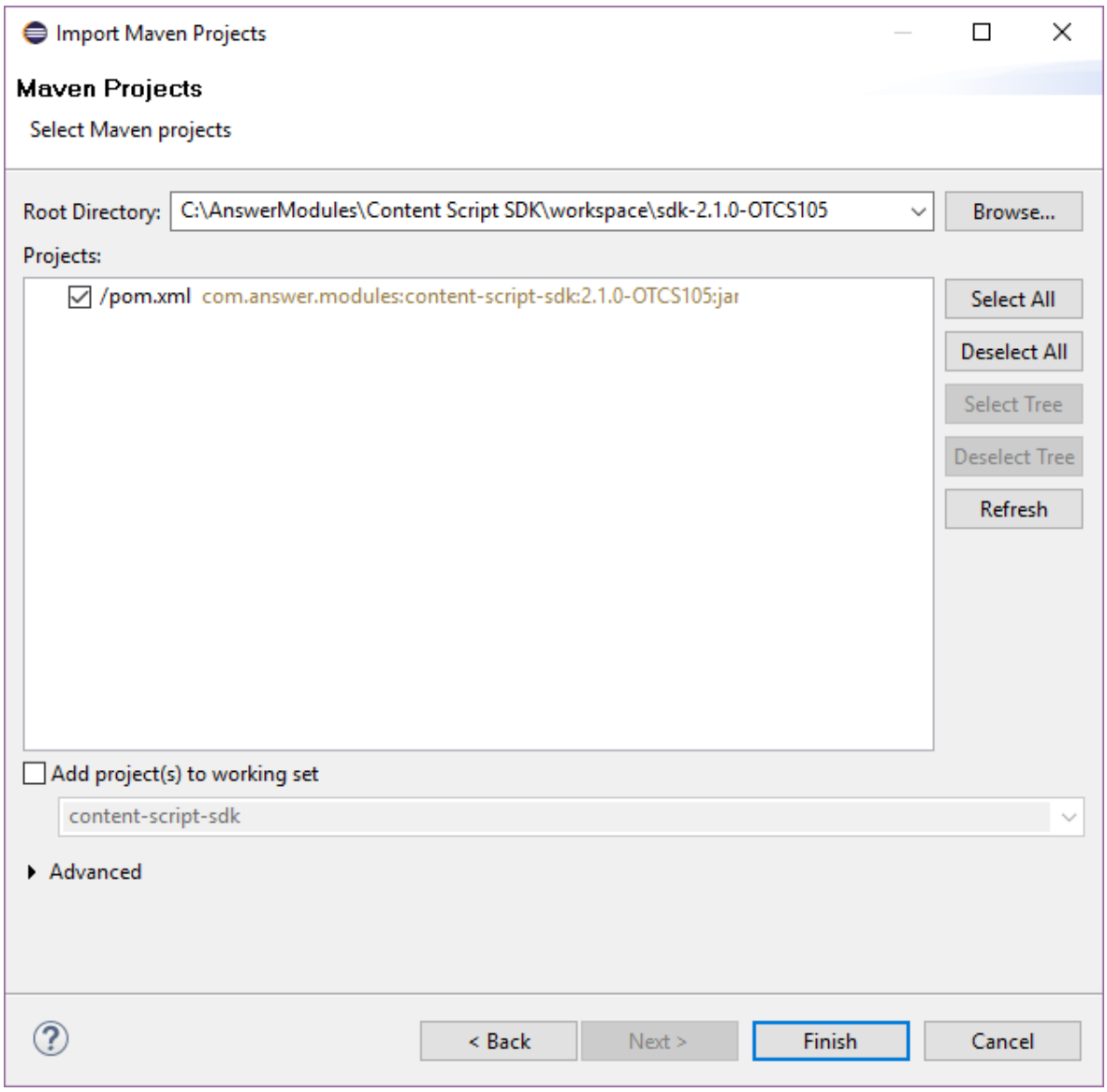
    2. install the components: **m2e - Maven integration for Eclipse**, **m2e - slfj over logback logging (Optional)**



4. In your workspace folder, unpack the contents of the **Content Script SDK** archive

5. Import the unpacked project within your new Eclipse environment.

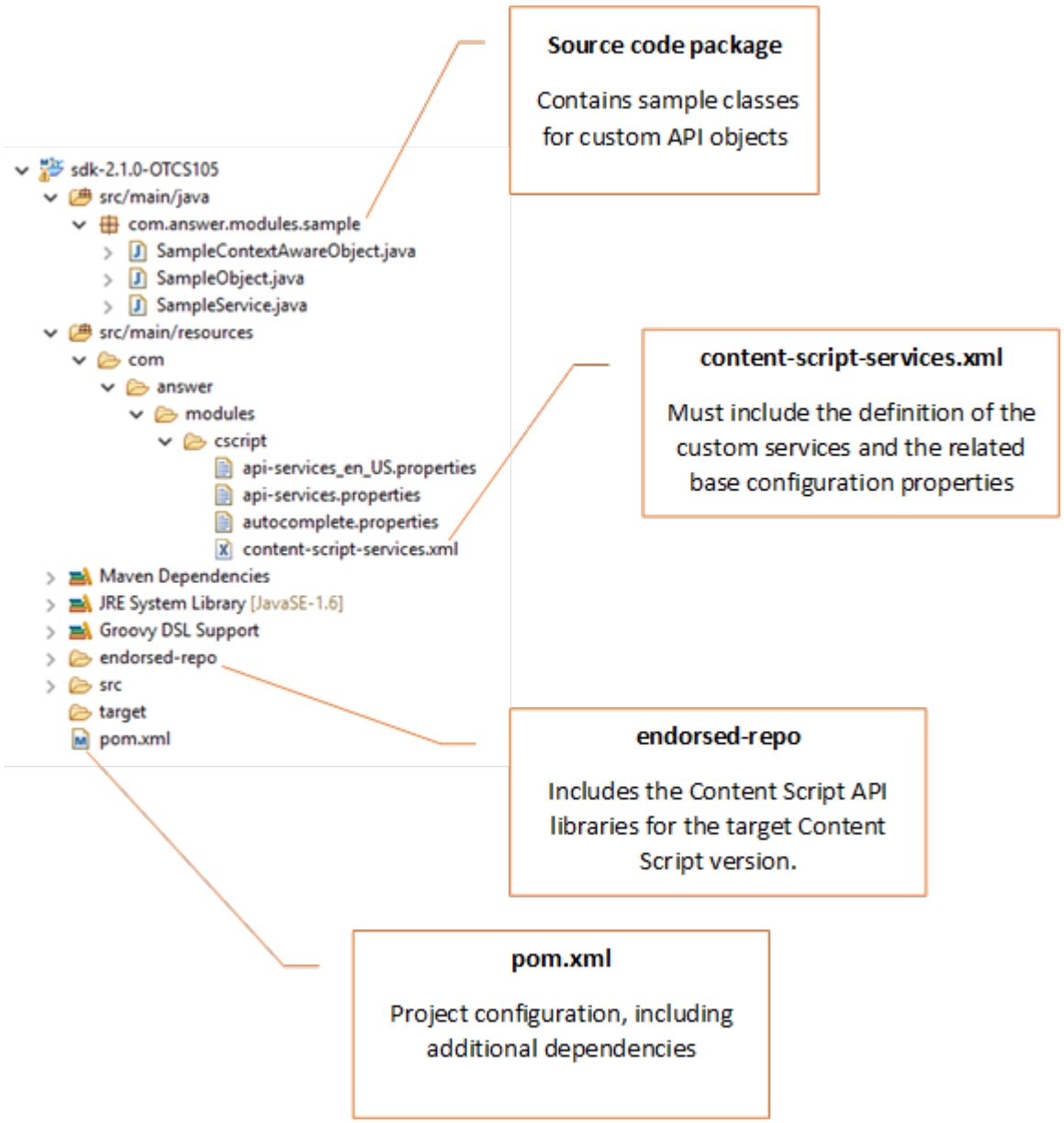




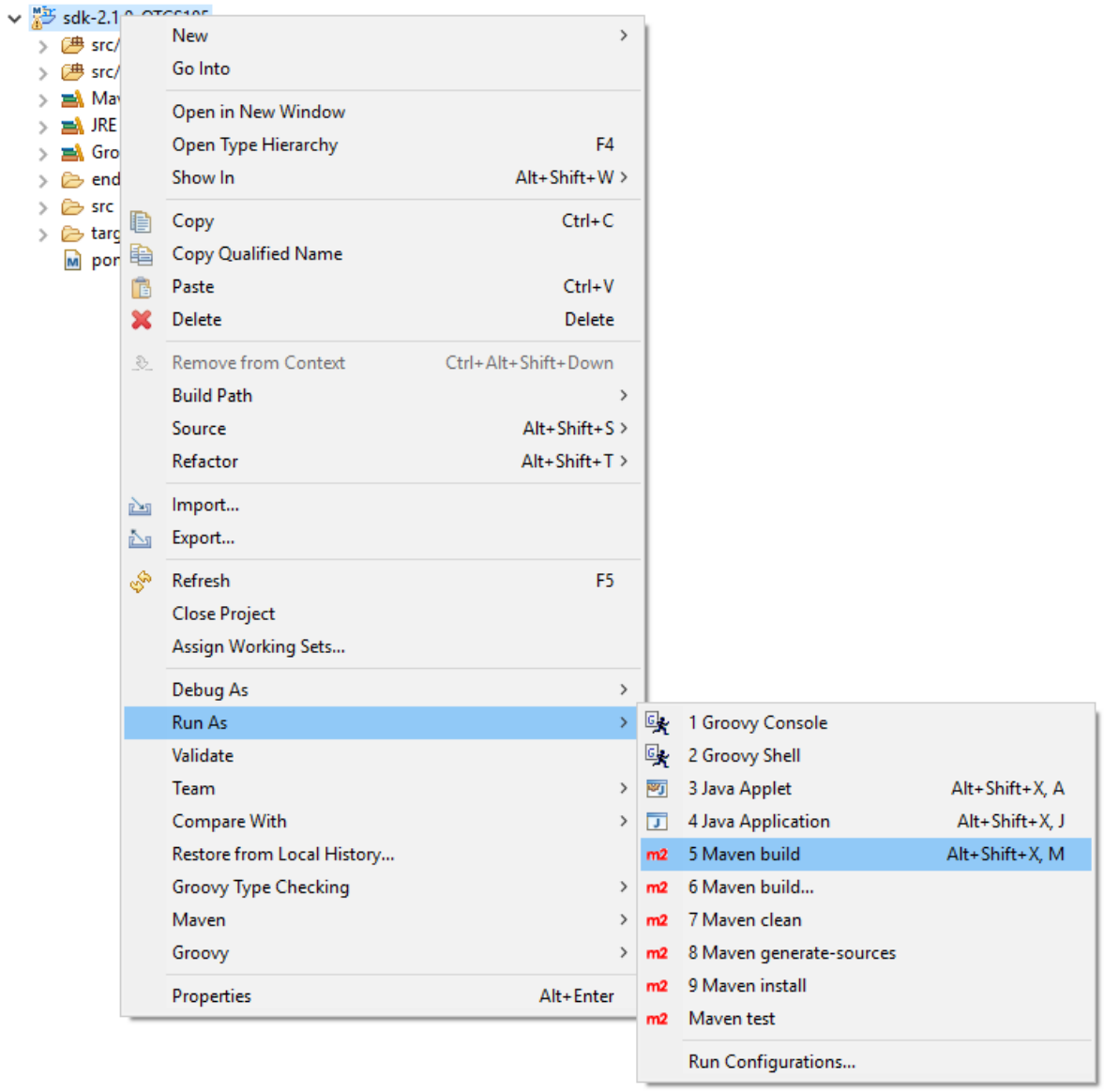
Navigate to the workspace folder and select the project directory, the project is identified by its pom.xml (Project Object Model) file. The Content Script SDK pom should appear in the listing.

Once selected, proceed with import.

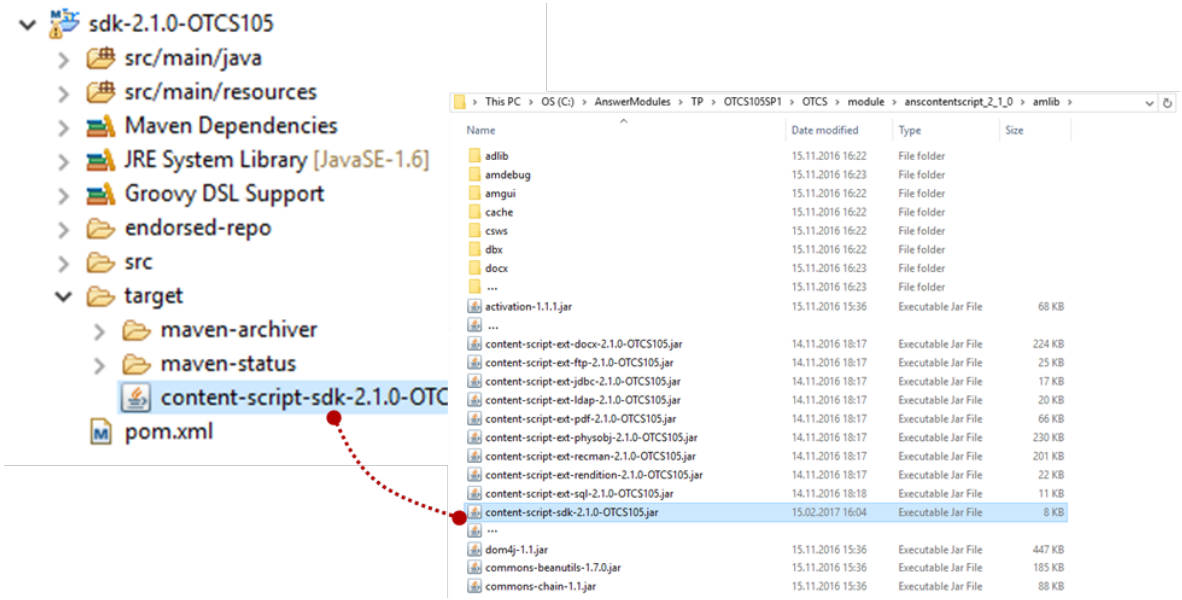
6. Review the imported SDK project layout



7. Build the project using the Maven menu options.



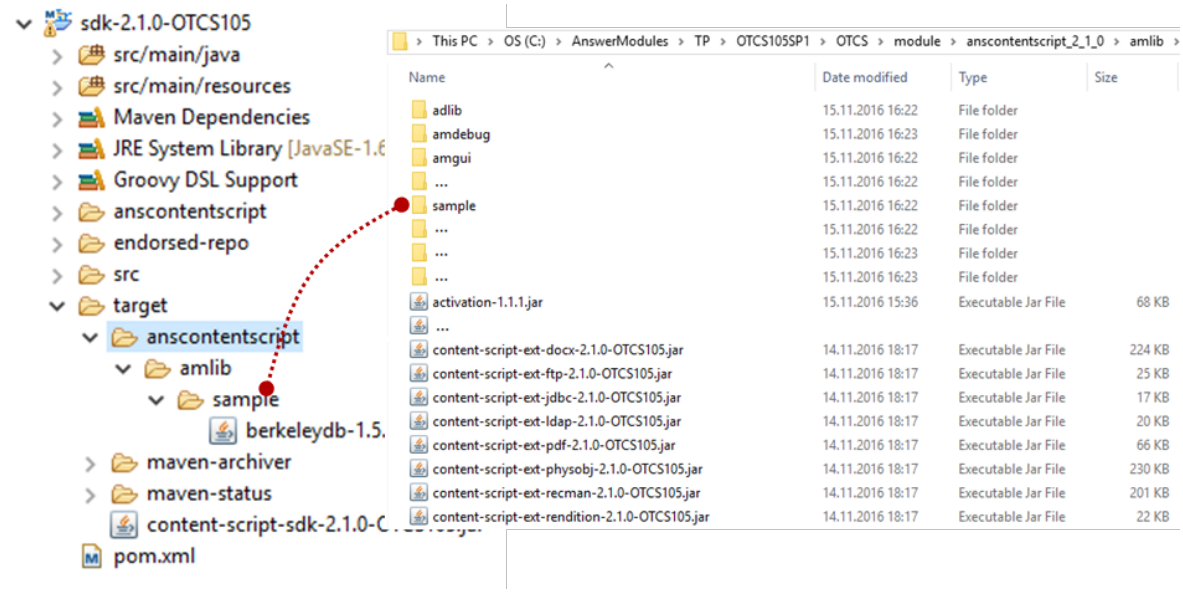
8. Deploy the newly created service on your Content Server instance. The main artifact produced by a project build is a jar file containing the service classes. In order to install the custom services to the target OTCS instance, copy the jar file to: **<OTCS_Home>/module/anscontentscript_X_Y_Z/amlib**



Each service might load as many dependencies as it needs, service's dependencies are loaded with an high isolation level, thus several services might load the same dependency (same library) or even load different version of the same dependency (different version of the same library). Service's dependencies are loaded by default from a folder stored under /module/anscontentscript_X_Y_Z/amlib having the same name as the service identifier. Service's dependencies can be specified using the POM file you can find in the SDK project. E.g.

```
<dependency>
    <groupId>berkeleydb</groupId>
    <artifactId>berkeleydb</artifactId>
    <version>1.5.1</version>
</dependency>
```

Upon build, an additional target folder will include all direct and indirect dependencies needed at runtime:



content-script-services.xml – Service description file¶

In order to let ModuleSuite be aware of your new service you have to properly describe it using the content-script-service.xml file. This xml files allows you not just to describe your service but also to provide some basic configuration for it.

The base structure of the file is as follows:

```
<?xml version="1.0" encoding="UTF-8" ?>
<services>
 <service id="sample" extRepoId="sample" class="com.answer.modules.sample.SampleService">
  <properties>
   <property name="sample.aProperty"
             description="A property with a default value (default: 'default')">default</property>
   <property name="sample.aSecret" type="hidden"
             description="A property with a hidden value"></property>
   <property name="sample.aNumber"
             description="A property with a numeric value (default: 1)">1</property>
  </properties>
 </service>
 <service id="anotherSample" extRepoId="sample" class="com.answer.modules.sample.ASampleService">
  <properties>
   <property name="sample.aProperty"
             description="A property with a default value (default: 'default')">default</property>
   <property name="sample.aSecret" type="hidden"
             description="A property with a hidden value"></property>
   <property name="sample.aNumber"
             description="A property with a numeric value (default: 1)">1</property>
  </properties>
 </service>
</services>
```

Using a single Content Scrip SDK project you can define as many services as you want. Each service should have its own service element descriptor in the description file. The mandatory attributes for the service element are: the service unique identifier (id) and the service implementation class (class). The extRepoId attribute is used if multiple services are defined in the same description file in order to inform ModuleSuite from where services' dependencies shall be loaded (in the above example both the services are loading their own dependencies from the same repository).

Content Script Extension for SAP¶

Using the extension¶

This section describes how to use the SAP API to retrieve data from the SAP system. The main Script API Object you are going to use is the SAPFunction object, which can be obtained from the sap service by calling sap.getFunction Script API Method. The SAPFunction object works the same for either an existing xECM connection or for a custom connection.

```
def sapfunc = sap.getFunction("BAPI_TIMEQUOTA_GETDETAILEDLIST", "PRD")
```

Function's input parameters can be specified using the **setImpParam** method:

```
def sapfunc = sap.getFunction("BAPI_TIMEQUOTA_GETDETAILEDLIST", "PRD")
sapfunc.setImpParam("EMPLOYEENUMBER", cid)
sapfunc.setImpParam("DEDUCTBEGIN", now)
sapfunc.setImpParam("DEDUCTEND", now)
```

To invoke a function in the target system and retrieve the function's result just call the **execute** method of the SAPFunction object:

```
def sapfunc = sap.getFunction("BAPI_TIMEQUOTA_GETDETAILEDLIST", "PRD")
sapfunc.setImpParam("EMPLOYEENUMBER", cid)
sapfunc.setImpParam("DEDUCTBEGIN", now)
sapfunc.setImpParam("DEDUCTEND", now)
sapfunc.execute()
```

Function execution results¶

The extension package features several options that help you in properly manage a function's execution result:

1. Function export parameter is in Table form Get content of table parameter of function execution result, i.e. as SapTable Script API Object. See sample code below

```
//result as SAPTable class
def sapTblQuote = sapfunct.table("ABSENCEQUOTARETURNTABLE",
                            "QUOTATYPE",
                            "QUOTATEXT",
                            "DEDUCTBEGIN",
                            "DEDUCTEND",
                            "ENTITLE",
                            "DEDUCT",
                            "ORDERED",
                            "REST",
                            "REST_FREE",
                            "TIMEUNIT_TEXT" )

def quote = sapTblQuote.rows.collect{
            [
                "quotaType":it.QUOTATYPE,
                "quotaText":it.QUOTATEXT,
                "begin":it.DEDUCTBEGIN,
                "end":it.DEDUCTEND,
                "entitle":it.ENTITLE,
                "deduct":it.DEDUCT+it.ORDERED,
                "rest":it.REST_FREE
            ]}
```

Please refer to SAPTable Script API Object for more detailed description of available methods and options.

1. Function export parameter is in Structure form Get content of a structure export parameter as a SapStructure Script API Object. See sample code below

```
def cumulateSAPStctr = sapfunct.table("CUMULATEDVALUES",
                                      "QUOTATYPE",
                                      "QUOTATEXT",
                                      "ENTITLE",
                                      "DEDUCT",
                                      "ORDERED",
                                      "REST",
                                      "REST_FREE",
                                      "TIMEUNIT_TEXT" )
//optionally you can call cumulateSAPStctr.getRows("QUOTATYPE","QUOTATEXT",...).collect()
def cumulate = cumulateSAPStctr.rows.collect{
                    [
                        "quotaType":it.QUOTATYPE,
                        "quotaText":it.QUOTATEXT,
                        "entitle":it.ENTITLE,
                        "deduct":it.DEDUCT+it.ORDERED,
                        "rest":it.REST_FREE
                    ]}
```

Please refer to SapStructure class API for more detailed description of available methods and options.

1. Get generic value of export parameter To get value of function export parameter you can use gertExportParam() method. Please see sample code below:

```
def empldet = sap.getFunction("Z_HR_MSD_RFC01_AD_EMPL_SINGLE", "PRD")
                    .setImpParam("I_PERNR", cid).execute()
                    .getExportParam("E_AD_EMPL")
```

All necessary conversions between Java and ABAP data types are done automatically.

Sample code listing below contains sample usage scenarios of SAP integration extension:

```
        // BAPI Function
getSAPHRData = {
    cid ->
    def now = new Date()
    def sapfunct = sap.getFunction("BAPI_TIMEQUOTA_GETDETAILEDLIST", "PRD")
                        .setImpParam("EMPLOYEENUMBER", cid)
                        .setImpParam("DEDUCTBEGIN", now)
                        .setImpParam("DEDUCTEND", now)
                        .execute()
    def quote = sapfunct.table("ABSENCEQUOTARETURNTABLE",
                            "QUOTATYPE",
                            "QUOTATEXT",
                            "DEDUCTBEGIN",
                            "DEDUCTEND",
                            "ENTITLE",
                            "DEDUCT",
                            "ORDERED",
                            "REST",
```

```
                                "REST_FREE",
                                "TIMEUNIT_TEXT" ).rows.collect{
            ["quotaType":it.QUOTATYPE, "quotaText":it.QUOTATEXT, "begin":it.DEDUCTBEGIN, "end":it.DEDUCT
        }
        def cumulate = sapfunct.table("CUMULATEDVALUES",
                                      "QUOTATYPE",
                                      "QUOTATEXT",
                                      "ENTITLE",
                                      "DEDUCT",
                                      "ORDERED",
                                      "REST",
                                      "REST_FREE",
                                      "TIMEUNIT_TEXT" ).rows.collect{
            ["quotaType":it.QUOTATYPE, "quotaText":it.QUOTATEXT, "entitle":it.ENTITLE, "deduct":it.DEDUC
        }
        return ["quote":quote, "cumulate":cumulate]
    }


    quotaMap = getSAPHRData(cid)


    out << template.evaluateTemplate("""

    <div>
        #@cstable(['Quote', 'Begin', 'End', 'Entitle','Deduction', 'Rest'] {  '':'' } {  '':'' })
            #foreach(\$row in \$quotaMap.quote)
                <tr>
                    <td>\$row.quotaText</td>
                    <td>\$date.format('dd.MM.yyyy', \$row.begin)</td>
                    <td>\$date.format('dd.MM.yyyy', \$row.end)</td>
                    <td>\$row.entitle</td>
                    <td>\$row.deduct</td>
                    <td>\$row.rest</td>
                </tr>
            #end
        #end
    </div>

    """
    )
```

SAP service APIs¶

Method Summary	
SapFunction	**getFunction**(String functionName, String destinationName) Get a SAP function for the specified destination
SapFunction	**getFunction**(String functionName) Get a SAP function for the default destination ('default')

API Objects¶

SapField¶

Method Summary	
SapField	**setValue**(Object value)

Method Summary
Set the field value

Field Summary	
Object	**value** Get the field value

SapFunction¶

Method Summary	
SapFunction	**disableExpParam**(String paramName) Disable an export param
SapFunction	**enableExpParam**(String paramName) Enable an export param
SapFunction	**execute**() Executes the SAP function.
Object	**getChangingParam**(String paramName) Get a changing param
Object	**getExportParam**(String paramName) Get an export param
Object	**getImportParam**(String paramName) Get an import param
SapFunction	**setImpParam**(String paramName, Object paramValue) Set the value of an import param
SapStructure	**structure**(String structureName, String[] fieldNames) Fetch the content of a structure export parameter
SapTable	**table**(String tableName, String[] columnNames) Fetch the content of a table parameter

SapStructure¶

Method Summary	
Map<String, Object>	**getRow**(String[] columns) Return the table content as a list of maps
SapStructure	**setColumns**(String[] columns) Set the table columns in the list of maps
SapStructure	**setColums**(String[] columns) Set the table columns in the list of maps
	setRow(Map<String, Object> values)

Method Summary	
SapStructure	Add a row and set the key/value mappings for the row

Field Summary	
Map<String, Object>	**row** Return the table content as a list of maps

SapTable¶

Method Summary	
SapTable	**addRow**(Map<String, Object> values) Add a row and set the key/value mappings for the row
List<Map<String, Object>>	**getRows**(String[] columns) Return the table content as a list of maps
SapTable	**setColumns**(String[] columns) Set the table colums in the list of maps
SapTable	**setColums**(String[] columns) Set the table columns in the list of maps

Field Summary	
List<Map<String, Object>>	**rows** Return the table content as a list of maps

Extension: Classic UI

Customize an object's functions menu: CSMenu¶

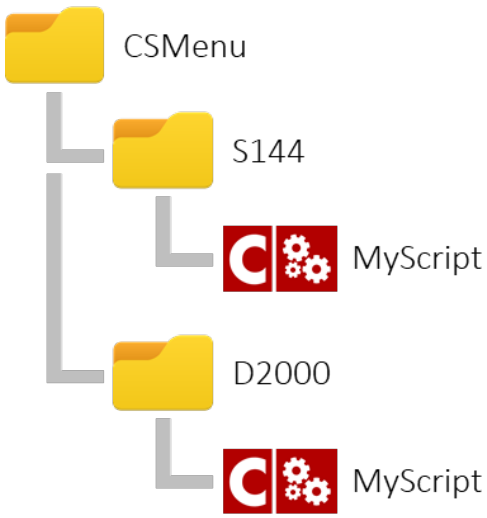
Content Script can be used to perform changes to the standard object function menus, by adding new options or removing existing ones. This feature is enabled by defining a Content Script that "filters" the object menu and performs the desired modifications. The "**amgui**" service provides a user-friendly interface to perform modifications to the menu object.

As for most other features configured through the Content Script Volume, a convention-over-configuration approach has been adopted.

The target container in which to place the Content Scripts is **CSMenu**. The first level under this container identifies the objects to which the customizations are applied. The naming convention is one of the following:

- `D<nodeID>`
- `S<subtype>`

where **nodeID** identifies the node unequivocally and **subtype** identifies a specific object subtype on Content Server.



Examples:

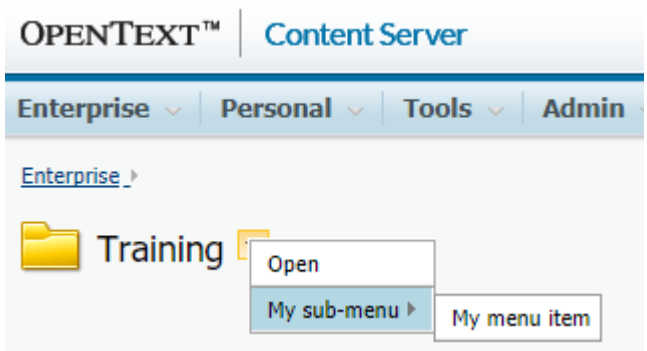
D2000 will change the function menu of the Enterprise Workspace

S144 will change the function menu of Document type objects (subtype: 144)

The following example shows a menu customization script that includes:

- fetching the original menu
- filtering the original menu entries (removing entries that match a specific expression)
- adding a divider row to split menu entries
- adding a submenu
- adding a custom menu entry to the new submenu
- returning the modified menu

E.g.



```
def csMenu = amgui.getCSMenu() //retrive the current object's menu
try{
    def node = docman.getNodeFast(nodeID)
    /**
    A filter is a closure that returns true if the menu item shall be kept, false otherwise.
    In the filter function scope the object "it" represent the menu item.
    A menu item has the following properties:
    - name (string)
    - url (string)
    - openInNewTab (boolean ) *available only on 10.5
    - order (decimal)
    **/
    csMenu.filter {it.name == "Open"}
    csMenu.appendDivider() //use appendDivider(position) to specify a position

    def submenu = csMenu.appendSubMenu("My sub-menu") //use appendSubMenu(name, position) to specify
        submenu.appendItem("My menu item", "${url}?func=ll&objAction=properties&objId=${nodeID}&next
}catch(e){
    log.debug("Unable to apply changes to add items menu",e)
}

return amgui.returnCSMenu(csMenu)
```

Property	Type	Description
name	String	Label of the menu entry (only for menu items and submenus)
openInNewTab	Boolean	If true opens a new target browser window (only for menu items)
position	String	The order of the entry in the menu (available for menu items, submenus and dividers)
url	String	The target URL (only for menu items)

> Notice that all operations are performed either through the **amgui** service or the **CSMenu** and **CSSubMenu** objects.

> **Return the proper value**
>
> The last operation performed in a CSMenu script should always be a call to the “returnCSMenu(...)” API of the amgui service

Customize a space's add-items menu: CSAddItems¶

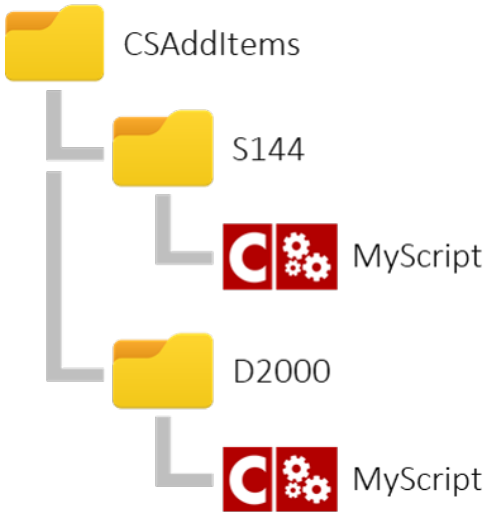
Content Script can be used to perform changes to a container’s Add Item menu, by adding new options or removing existing ones. This feature is enabled by defining a Content Script that “filters” the menu and performs the desired modifications. The “**amgui**” service provides a user-friendly interface to perform modifications to the menu object.

As for most other features configured through the Content Script Volume, a convention-over-configuration approach has been adopted.

The target container in which to place the Content Scripts is **CSAddItems**. The first level under this container identifies the objects to which the customizations are applied. The naming convention is one of the following:

- `D<nodeID>`
- `S<subtype>`

where **nodeID** identifies the node unequivocally and **subtype** identifies a specific object subtype on Content Server.



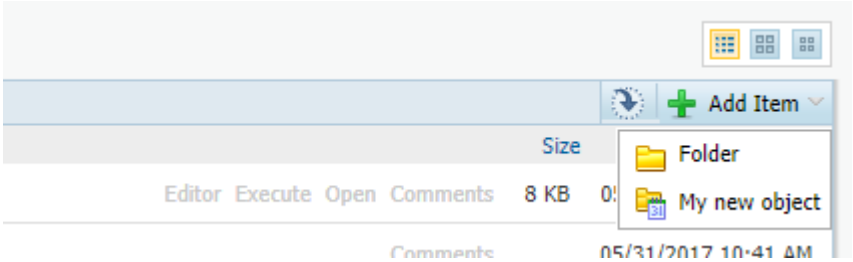
Examples:

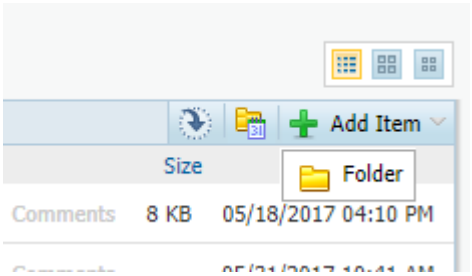
D2000 will change the add items menu of the Enterprise Workspace

The following example shows a menu customization script that includes:

- filtering the original menu entries (removing entries that match a specific expression)
- adding a custom menu entry
- returning the modified menu

E.g.






```
try{
    //The current space
    def node = docman.getNodeFast(nodeID)
    /**
        Other possibile filter examples:
        it.name == "Folder"
        it.subtype == 0
    **/
    amgui.filterAddItems {
        it.name == "Folder"
    }
    /**
        Other possibile filter examples:
        it.name  == "Folder"
        it.subtype == 0
    **/
    amgui.filterAddItems ({false}, true)
    amgui.addBrowseViewAddItem(
        amgui.newBrowseViewAddItemsMenu().builderUrl().setImg("${img}folder_icons/folder5.gif")
                                                      .setName("My new object")
                                                      .setPromoted(true)
                                                      .setUrl("${url}?func=ll&objAction=create&objTy
    )
}catch(e){
    log.debug("Unable to apply changes to add items menu",e)
}

return amgui.returnAddItemsMenus()
```

Invoke a Content Script

The url of the menu entry could be used to pass parameters to a custom Content Script that will perform the desired operations.

Return the proper value

The last operation performed in a CSAddItems script should always be a call to the “returnAddItemsMenu(...)” API of the amgui service

Customize a space's buttons bar: CSMultiButtons¶

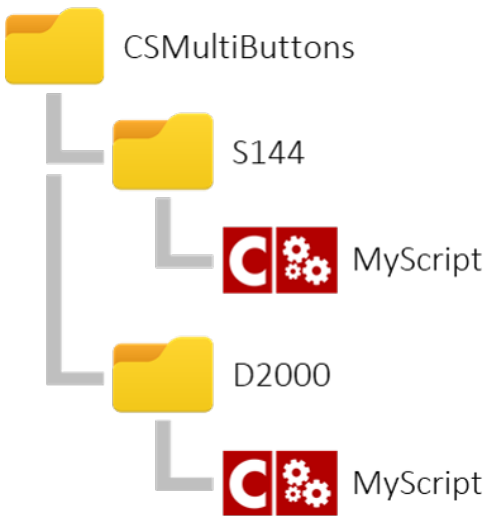
Multi-action buttons can be added, removed or modified by using an approach similar to the CSMenu customization. In this case, customization scripts should be added in the **CSMultiButtons** container. The container structure is the same as the one described for the CSMenu.

As for most other features configured through the Content Script Volume, a convention-over-configuration approach has been adopted.

The target container in which to place the Content Scripts is **CSMultiButtons**. The first level under this container identifies the objects to which the customizations are applied. The naming convention is one of the following:

- `D<nodeID>`
- `S<subtype>`

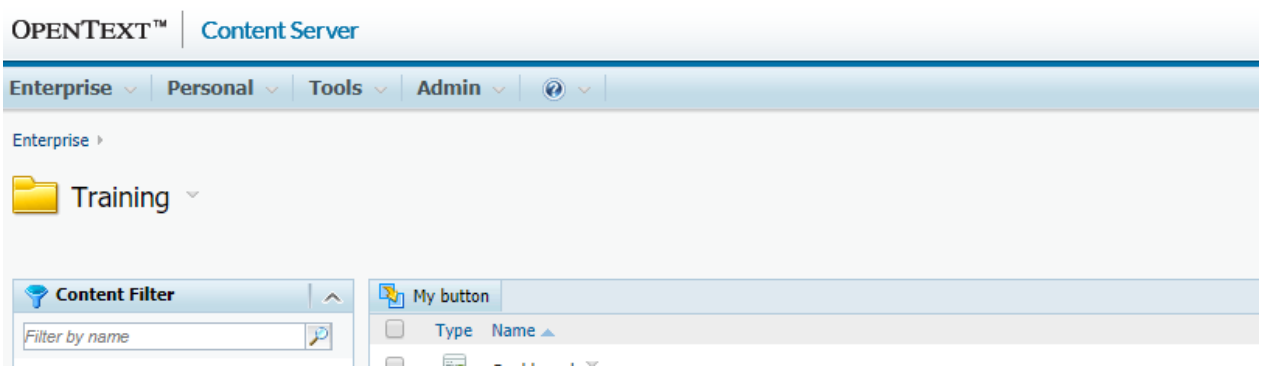
where **nodeID** identifies the node unequivocally and **subtype** identifies a specific object subtype on Content Server.



Examples:

D2000 will change the buttons bar menu of the Enterprise Workspace

E.g.




```
try{
    amgui.addBrowseViewMultiItemButton(
        amgui.newBrowseViewMultiItemButton()
            .builder()
             .setOrder(1100)
             .setJavascriptFunctionName('runContentScript')
             .setJavascriptFile("anscontentscript/js/contentScriptMultifileBar.js")
             .setImageMap("anscontentscript/contentscriptmultifilebar.png")
             .setImageXPos(0)
             .setImageYPos(0)
             .setImageXPosAlternative(-268)
             .setImageYPosAlternative(0)
             .setDisplayName('My button')
             .create()
    )
    /**
        Properties that can be used to filter the buttons bar:
        - action (the request handler to be executed e.g. ll.ProcessMultiCopy)
        - Order
        - Name
        - DisplayName
        - ExecutesOnClient

    **/
    amgui.filterBrowseViewMultiItemButton {it.name == "mybutton"}

}catch(e){
    log.debug("Unable to apply changes to add multi items buttons bar",e)
}
return amgui.returnBrowseViewMultiItemButtons()
```

where the following fields are mostly relevant:

Property	Type	Description
ImageMap	String	The path of the image map file (in the Support folder) containing the button icon
ImageXPos, ImageXPos2, ImageYPos, ImageYPos2	Integer	The coordinates of the portion of the image map to use for the button (normal and on mouse over)
Order	String	The order of the button in the menu bar
Type	String	The button type (should be “Content Script”)
ExecutesOnClient	boolean	

Property	Type	Description
		If the button logic is on the client side (should be “true”)
DisplayName	String	The button label
Name	String	The name of the button
JavascriptFile	String	The javascript resource in which the function controlling the button behavior is defined
JavascriptFunctionName	String	The javascript function defined in the JavascriptFile that controls the button behavior

Invoke a Content Script

A sample Javascript file (contentScriptMultifileBar.js) is located in the Content Script Module support folder. Create a customized version of this file when adding new actions.

Customize a space's displayed columns:

CSBrowseViewColumns¶

Content Scripts located in the **CSBrowseViewColumns** container can be used to perform modifications to how columns are presented in the standard Content Server Browse View.

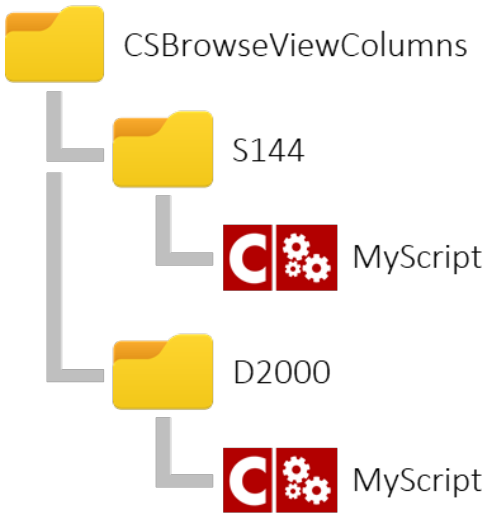
The modifications can be limited to specific portions of Content Server. This feature is enabled by defining a Content Script that “filters” the browse view columns configuration and performs the desired modifications. The “**amgui**” service provides a user-friendly interface to perform the modifications.

As for most other features configured through the Content Script Volume, a convention-over-configuration approach has been adopted.

The target container in which to place the Content Scripts is **CSBrowseViewColumns**. The first level under this container identifies the objects to which the customizations are applied. The naming convention is one of the following:

- `D<nodeID>`
- `S<subtype>`

where **nodeID** identifies the node unequivocally and **subtype** identifies a specific object subtype on Content Server.



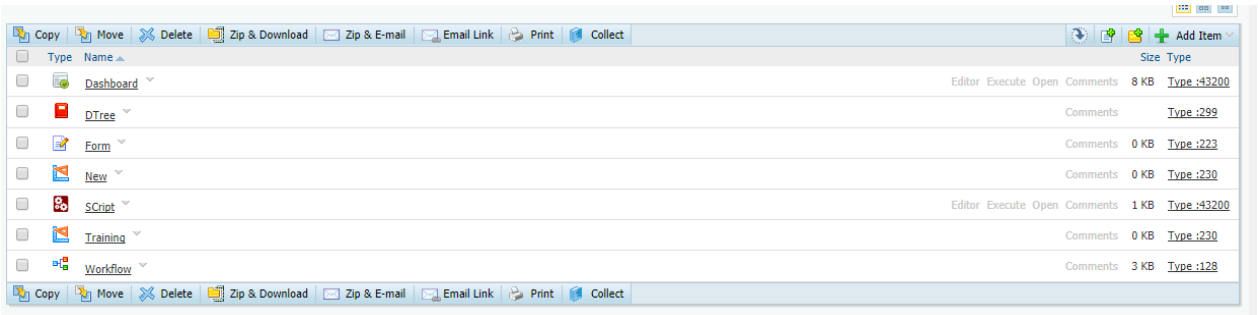
Examples:

D2000 will change the columns visible in the Enterprise Workspace

The following example shows a browse view columns customization script that includes:

- create a new column using the builder
- filtering the original columns list (removing entries that match a specific expression)
- adding the column to the view
- returning the modified columns list

E.g.




```
try{
    /**
    A browse view column is quite a complex object. The amgui service provides you with a builder in
    **/
    def columnBuilder = amgui.newBrowseViewColumn().builder()

        .setColumnName("type")  // Column name corresponds to the property
                                //from the browse view row that will be used
                                //to populate the column.
        .setDisplayName("Type") // Column display name is the label used for the column.
        .setAlignment("left")
        .setSortable(true)      // If sortable the Javascript sorting
                                //function will look for a property named:
```

```
                                // columnName+'SortStr' or columnID+'SortStr'
                                // to perform sorting
        .setColumnEMWidth(1.0)
        .setDisplayAsLink(true)
        .setNewWindow(true)
        .setUrl("${url}?param=%value%")      // The url to be opened.
                                             // The following placeholder
                                             // can be used in the expression:
                                             // %value%, %objid%, %rawvalue%, %nexturl%"

        .setFormatValueMask("Type :%value%") // The format mask to be used to
                                             // present the column value.
                                             // The following placeholder
                                             // can be used in the expression:
                                             // %value%, %objid%, %rawvalue%, %nexturl%"
    /**
    A filter is a closure that returns true if the column shall be kept, false otherwise.
    In the filter function scope the object "it" represent the column object.
    For default columns the only attribute available is columnID (string) which might have one out t
    dataidColumn, dateColumn, arbitraryColumn, columnWithURL, userColumnWithURL)

    All the other columns have the following properties:
    DisplayAsLink (boolean), DisplayValue (string), NewWindow (boolean), NewWindowTitle (string), UR

    **/
    amgui.filterBrowseViewColumn {
        it.columnID != "dateColumn"
    }
    amgui.addBrowseViewColumn(columnBuilder.create())

}catch(e){
    log.debug("Unable to apply changes to add items menu",e)
}

return amgui.returnBrowseViewColumns()
```

The following properties are available for each column object (they are managed through a builder *(https://en.wikipedia.org/wiki/Builder_pattern)* the `CSBrowseViewColumnBuilder` obtained :

Property	Type	Description
isDefault	boolean	True if the column has a Javascript definition
sortable	boolean	True if the column has a Javascript definition
DisplayAsLink	boolean	The value of the column will be wrapped into an HTML link
DisplayValue	String	The column's value
NewWindow	boolean	If DisplayAsLink = true, opens the link in a new window
NewWindowTitle	String	If DisplayAsLink = true, the title of the window in which link will be opened
Url	String	If DisplayAsLink = true, the URL to be used for building the link
alignment	String	Column alignement. One out: 'left', 'right', 'center'
columnID	String	Column unique identifier
columnName	String	Column name
displayName	String	Column name as it will be displayed in the page

Property	Type	Description
displayName	String	Column name as it will be displayed in the page

Filtering columns - lines from 39 to 41

A filter is a closure that returns true if the column shall be kept, false otherwise.

Default Columns¶

Default columns are columns for which a Javascript column definition exists. Default columns Javascript definitions can be found in webnode/browse.js file. The following **default** columns definition should exist in your environment:

Value	Description
checkBoxColumn	Used for selecting multiple nodes
typeColumn	Represents the node's type in the form of a web-icon
nameWthPrmtdCmdsColumn	Name with promoted commands column
sizeColumn	Size of the document or number of items in the space
dataidColumn	Node's unique system identifier
dateColumn	Node's last modification date
arbitraryColumn	Template for other columns (ABSTRACT)
columnWithURL	Template for other columns (ABSTRACT)
userColumnWithURL	Node's owner

The amgui service features a method that can help you in creating your own custom column Javascript definition on the basis of a template that is stored in the Content Script Volume (`CSVolume:CSGui:BrowseViewColumnDefinition`). The custom Javascript column's definition can be rendered, for example, as part of a customview, an appearance or a Content Script

```
amgui.getBrowseViewColumnDefinition(
                                    String columnID, //The id of the column
                                    Map templateContext, // A map to be used as model for
                                                         // the column's definition template
                                    [,CSDocument param ] // An optional template document.
                                                         // If none is provided the default
                                                         // CSVolume:CSGui:BrowseViewColumnDefinitio
                                                         // will be used
                                    )
```

Here below a real-world usage example. The Script is used to create a custom view within the space in which is stored.

```
jsAddCell = """
        var cell;

        try
        {

            cell = rowStruct.insertCell( cellCount++ );
            cell.className = this.cellClassName;
            if ( true === this.nowrap )
            {
                cell.style.whiteSpace = 'nowrap';
            }
            cell.innerHTML = this.getCellValue( dataRow, rowNo );


        }
        catch(e)
        {
            exceptionAlert( e, "Issue occured in browse.js/htmlColumn.AddCell." );
        }
        return cellCount;
"""

jsGetCellValue ="""
    var val = dataRow[ 'pstatus' ];
    if ( val == undefined )
    {
        val = "";
    }
    return val;
"""

def customView = docman.getTempResource("customView", ".html")

customView.content.withWriter{
    it << amgui.getBrowseViewColumnDefinition("pstatus",
                                              ["jsAddCell":jsAddCell, "name":"Status", "jsGetCellVal
}
def cv = docman.createCustomView(self.parent, "customView", customView.content)
cv.setIsHidden()
cv.update()
```

For **default** columns (listed in the table above) the only attribute available is **columnID** (string).

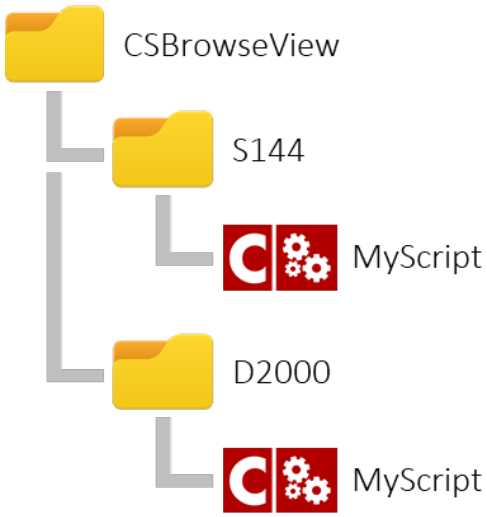
Customize a space content view: CSBrowseView¶

Content Scripts located in the **CSBrowseView** container can be used to perform modifications on the content of a browse view.

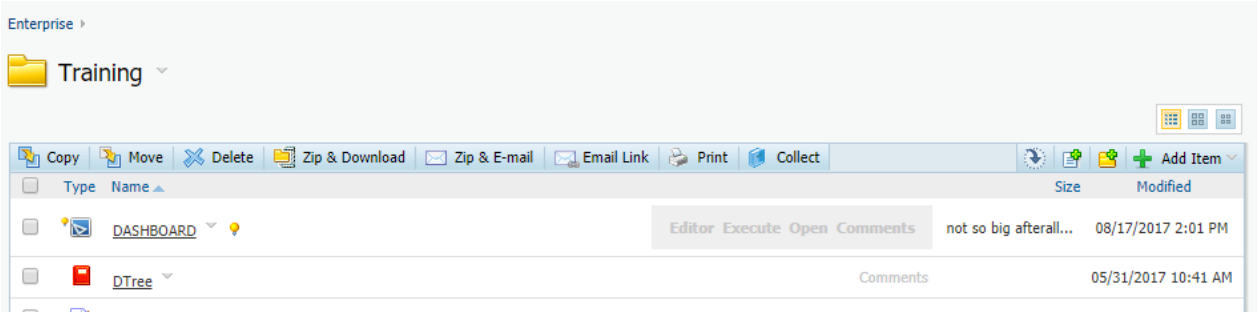
The target container in which to place the Content Scripts is **CSBrowseView**. The first level under this container identifies the objects to which the customizations are applied. The naming convention is one of the following:

- `D<nodeID>`
- `S<subtype>`

where **nodeID** identifies the node unequivocally and **subtype** identifies a specific object subtype on Content Server.



The following example shows a browse view customization script that will iterate on each row in the browse view and perform modifications for objects of subtype 43200 (Content Scripts)




```
try{

    /**
        Properties that can be used to filter the browse view rows:
        - dataId (Numeric)
        - name (String/Html will be rendered inside an 'a' tag)
        - link (String)
        - size (String/Html e.g. '1 KB')
        - date (String e.g.
        - imgStr (String)
        - imgLargeStr (String)
        - imgThumbnailStr
        - promotedCmds (Html)
        - modifiedImgs (Html)
        - imgStatus (String)
        - statusName (String)


    **/
    amgui.filterBrowseView { row ->

        // Just for Content Scripts
        if(row.type == "43200"){
            row.checked = true
            row.name = "${row.name.toUpperCase()}"
            row.promotedCmds = """ <div style="font-weight:bold;background-color:#EFEFEF;padding:10p
            row.modifiedImgs =  "<img src='${img}webnode/new.gif' />"

            row.imgStatus  = "${img}webnode/new.gif"
            row.statusName = "Ready to be executed"
```

```
            row.link ="http://www.answermodules.com/products/content-script"
            row.size = "not so big afterall..."

            row.date = amgui.formatDateForBrowseView(new Date())  //This is a shortcut to format dat

            row.imgStr = "${img}anscontentscript/lib/img/icons/product-design.png"
            row.imgLargeStr = "${img}anscontentscript/lib/img/icons/product-design_large.png"
            row.imgThumbnailStr = "http://www.answermodules.com/img/content-script/content-script-ba
        }

        // This to be sure that the rows will be rendered
        return true
    }
}catch(e){
    log.debug("Unable to filter browse view rows for node {}", nodeID, e)
}
return amgui.returnBrowseViewRows()
```

Filtering rows - lines from 20 to 42

The filtering closure passed as parameter to the amgui.filterBrowseView(...) method should return a boolean value of “true”. If “false”, the row will not be rendered.

Add a new row

It is possible to add new rows from scratch by using the amgui.addBrowseViewRow(...) method. A blank row template can be obtained through the amgui.newBrowseViewRow() method

The following properties are available for filtering or modification on each **row** object that is being iterated:

Property	Type	Description
dataId	Numeric	The node's unique identifier
name	String/ HTML	The node's name Html will be rendered inside an 'a' tag
link	String	The link to be associated to the node's name
size	String/ HTML	The node's side e.g. '1 KB'
date	String	The node's last modification date
imgStr	String	The url for the node's icon
imgLargeStr	String	The url for the node's icon when the node is featured
imgThumbnailStr	String	The url for the node's thumbnail
promotedCmds	HTML	The HTML code containing links to the node's promoted functions (can be any HTML)
modifiedImgs	HTML	The HTML code to be used to notify users that the node's has been modified

Property	Type	Description
imgStatus	String	The url for the node's status icon
statusName	String	The node's status name

Create a custom column backed by Content Script: CSDataSources¶

Since version 1.5 Content Scripts can be used as Column Data sources. Content Scripts placed in the **CSDataSources** Template Folder will automatically be available as Column Data Sources.

The CSDataSource scripts will automatically be invoked by Content Server for each node of the system, and the resulting value will be used as a column value.

Return the proper value

A CSDataSource Content Script **MUST** always return a String object.

In the Content Script code, the execution context will be enriched by the framework with the following information related to the current node:

- volumeID
- parentID
- dataID
- createDate
- modifyDate

As per standard column data sources the developer is in charge of defining and implementing a reliable updating strategy. Most of the time the task can be accomplished implementing either a synchronous or an asynchronous (see Managing events) event script.

As a matter of fact, Content Script features two different APIs that can be used to update columns' datasources values.

```
docman.updateColumnValue( CSNode node, //The node for which you want to update the column's value
                          String dataSourceId, // The standard identifier for the column's datasourc
                          String columnValue // The new value for the column
                         )

docman.updateContentScriptColumnValue( CSNode node, //The node for which you want to update the colu
                          String scriptName, // The name of the Content Script script that serves
                                             // datasource
                          String columnValue // The new value for the column
                         )
```

The first one is supposed to be used with standard columns' datasources, the latter with Content Script backed columns' datasources.

The **updateContentScriptColumnValue** takes as secondo parameter the name of the Script used to implement the column's datadasouce.

The **updateColumnValue** method takes as second parameter a **dataSourceIdentifier**, which can be easily determined inspecting the ExtendedData column's value of the corresponding Column object on the DTree table (property "**dataSource**").

E.g.

```
def exData = sql.runSQL( """ select ExtendedData EXT
                             from   DTree
                             where  DataId = %1 """,
                         false,
                         false,
                         -1,
                         2109 //The column object DataId
                        ).rows[0].EXT
out << extData.getMapFromOscript().dataSource //Returns sys_CreateDate
                                              //(on most of the systems)
```

Extension: AI (LLM)

CARL OpenAI llm

LLM services

Required Hotfix for Version 3.7.0

It has been determined that hotfix `hotFix_ANS_370_004` is required for the LLM service to work properly on Module Suite version 3.7.0. Please ensure this hotfix is applied to your system before using the LLM service. Failure to apply this hotfix may result in unexpected behavior or errors when using LLM-related features.

To obtain and apply this hotfix, please contact AnswerModules support or refer to the official documentation for hotfix installation procedures.

Integrate Large Language Models in your workflow¶

Introduction¶

Large Language Models (LLMs) are revolutionizing the way organizations process and leverage information. These sophisticated AI models, trained on vast amounts of textual data, can understand, generate, and manipulate human-like text with remarkable accuracy. As businesses increasingly deal with enormous volumes of unstructured data, integrating LLMs into existing workflows has become a game-changer for enhancing productivity, improving decision-making processes, and unlocking new insights.

Module Suite plays a crucial role in seamlessly incorporating LLMs into your enterprise content management ecosystem. By bridging the gap between your organization's content repositories and cutting-edge AI capabilities, Module Suite empowers you to:

1. **Automate content classification**: Leverage LLMs to automatically categorize and classify documents, making information retrieval faster and more accurate.
2. **Enhance search functionality**: Utilize natural language processing to improve search results, allowing users to find relevant information using conversational queries.
3. **Generate intelligent summaries**: Create concise summaries of lengthy documents, enabling quick understanding of key points without manual review.
4. **Streamline content creation**: Assist users in drafting documents, emails, and reports by providing AI-powered suggestions and completions.

5. **Facilitate knowledge discovery**: Uncover hidden patterns and relationships within your content, leading to valuable insights for decision-makers.

6. **Improve data extraction**: Extract relevant information from unstructured documents, making it easier to populate structured databases or forms.

By integrating LLMs through Module Suite, organizations can harness the power of AI to transform their content management processes, leading to increased efficiency, reduced manual effort, and improved overall productivity.

> **LLM Integration Considerations**
>
> While LLMs offer significant benefits, it's important to consider factors such as data privacy, model selection, and fine-tuning requirements when integrating them into your workflow. Module Suite provides the necessary tools and interfaces to address these considerations effectively.

Architecture and Networking¶

Module Suite acts as a central hub for communication between Extended ECM (xECM), various LLM API services, and local resources.

Here's an overview of how the networking and communication work:

```
flowchart TD
    subgraph ECM["Extended ECM (xECM)"]
        MS[Module Suite]
    end

    API1[OpenAI API]
    API2[Azure AI API]
    API3[Ollama API]

    MS <--> |Internal APIs| ECM
    MS <--> |HTTP/REST| API1
    MS <--> |HTTP/REST| API2
    MS <--> |HTTP/REST| API3

    style ECM fill:#f9f,stroke:#333,stroke-width:2px
    style MS fill:#bbf,stroke:#333,stroke-width:2px
    style API1 fill:#bfb,stroke:#333,stroke-width:2px
    style API2 fill:#bfb,stroke:#333,stroke-width:2px
    style API3 fill:#bfb,stroke:#333,stroke-width:2px
```

Integration with xECM¶

Module Suite runs directly on xECM, providing seamless access to all xECM APIs. This tight integration allows for efficient data exchange and leveraging of xECM's content management capabilities.

LLM API Communication¶

Module Suite implements independent communication channels to various LLM API providers, which can be: - Public internet services (e.g., OpenAI) - VPN-accessible services (e.g., Azure AI) - On-premises solutions (e.g., LLAMA3 using Ollama to expose an API)

OpenAI-Compatible API Privileged Support

Module Suite features a rich API specifically designed for OpenAI-compatible API service providers, most commonly used with OpenAI and Azure. This allows for flexible integration with different LLM services while maintaining a consistent interface.

Local Embedding Indexes¶

To enhance performance and maintain data control, Module Suite allows administrators to configure and create local embedding indexes. These indexes are typically used for implementing Retrieval-Augmented Generation (RAG) systems. Key points include:

- Based on an adapted version of the Lucene open-source indexing engine
- Requires sending text chunks to the LLM API service provider for embedding computation
- Does not store entire documents outside your organization
- Provides full control over chunking policies and methodologies

Permission Considerations

When implementing local embedding indexes, it's crucial to ensure that permissions are properly considered. This helps maintain data security and access control in line with your organization's policies. We will explore this in detail in the following sections.

Typical Communication Sequence¶

Below is a diagram illustrating a typical communication sequence when using Module Suite with xECM and an LLM API service for implementing a RAG:

```
sequenceDiagram
    participant User
    participant xECM
    participant Module Suite UI Widget
    participant Module Suite (Script Engine)
    participant LocalIndex
```

```
participant LLMAPI


User->>xECM: Request content
xECM->>Module Suite UI Widget: Pass request
Module Suite UI Widget->>Module Suite (Script Engine): Pass request and history
Module Suite (Script Engine)->>LocalIndex: Query local index
LocalIndex-->>Module Suite (Script Engine): Return relevant chunks coordinates
Module Suite (Script Engine)->>xECM: Retrive relevant chunks (context)
xECM-->>Module Suite (Script Engine): Returns relevant chunks
Module Suite (Script Engine)->>LLMAPI: Send prompt with context
LLMAPI-->>Module Suite (Script Engine): Return LLM response
Module Suite (Script Engine)->>Module Suite UI Widget: Process and format response
Module Suite UI Widget->>xECM: Render result
xECM->>User: Display result
```

Service Provider Support in Module Suite¶

Module Suite offers extensive support for various LLM API providers, with a focus on OpenAI-compatible APIs and limited support for other providers. Below is a detailed breakdown of the supported features for each provider type.

OpenAI API Providers (OpenAI and Microsoft Azure AI)¶

Module Suite provides comprehensive support for OpenAI-compatible APIs, including those from OpenAI itself and Microsoft Azure AI. The following features are supported:

1. Chat Completion
2. Text Completion
3. Function Invocation
4. Vision (image analysis and processing)
5. Text-to-Speech
6. Speech-to-Text
7. Assistant API
8. Embeddings
9. Fine-tuning
10. Moderation

This wide range of supported features allows for versatile integration of AI capabilities into your Extended ECM workflows, enabling tasks from simple text generation to complex multimodal interactions.

Ollama API Support¶

For Ollama-based API providers, Module Suite currently offers limited but essential support:

1. Embeddings
2. Chat Completion

While more restricted than the OpenAI API support, these features still allow for crucial functionalities such as semantic search and conversational AI interactions using on-premises or self-hosted models.

```
graph TD
    A[Module Suite API Support] --> B[OpenAI APIs]
    A --> C[Ollama APIs]

    B --> D[Chat Completion]
    B --> E[Text Completion]
    B --> F[Function Invocation]
    B --> G[Vision]
    B --> H[Text-to-Speech]
    B --> I[Speech-to-Text]
    B --> J[Assistant API]
    B --> K[Embeddings]
    B --> L[Fine-tuning]
    B --> M[Moderation]

    C --> N[Embeddings]
    C --> O[Chat Completion]

    style B fill:#f9f,stroke:#333,stroke-width:2px
    style C fill:#bbf,stroke:#333,stroke-width:2px
    style N stroke:#333,stroke-width:2px
    style O stroke:#333,stroke-width:2px
```

API Support Evolution

The landscape of LLM APIs is rapidly evolving. Module Suite's API support is regularly updated to include new providers and features. Always refer to the latest documentation for the most up-to-date information on supported APIs and features.

Choosing the Right API Provider

When selecting an API provider for your Module Suite implementation, consider the following factors:

- **Feature requirements**: Assess which AI capabilities are crucial for your use case.
- **Data privacy and compliance**: Determine if you need to keep data on-premises or if cloud-based solutions are acceptable.

- **Performance needs**: Evaluate the response times and throughput required for your applications.
- **Cost considerations**: Compare pricing models of different providers, especially for high-volume usage.
- **Integration complexity**: Consider the ease of integration with your existing infrastructure.

Components of the LLM Service Integration¶

Module Suite provides a comprehensive set of components to enable seamless integration with LLM services. These components work together to offer a robust and flexible AI-enhanced experience within the Extended ECM environment. Let's explore each of these components:

Content Script¶

Module Suite features a set of dedicated extension package to support LLM API integration.

OpenAI Extension Package Service¶

The OpenAI service is a dedicated Content Script extension package service specifically designed for OpenAI-compatible API integrations. Key features include:

- Multi-profile support for flexible configuration
- Comprehensive implementation of OpenAI's API features
- Optimized for use with OpenAI and Azure AI services

LLM Extension Package Service¶

The LLM service is a more general-purpose Content Script extension package service for integrating various LLM providers. Its features include:

- Multi-profile configuration for supporting different LLM services
- Currently focused on Ollama API support
- Extensible architecture for future LLM provider integrations

Widgets¶

Smart Pages Widget (named CARL)¶

This Smart Pages widget brings AI-powered capabilities directly into the Extended ECM user interface. This widget:

- Provides an interactive AI assistant interface within Smart Pages
- Leverages the power of LLM models for various tasks
- Enhances user productivity by offering AI-assisted functionalities

Beautiful WebForm Widget (named CARL)¶

This Beautiful WebForm Widget extends AI capabilities to WebForms, allowing for:

- AI-enhanced form interactions
- Intelligent form filling assistance
- Dynamic content generation based on form context

Services¶

Content Script Service (named carl)¶

This Content Script Service acts as the backend engine for LLM-related functionalities, providing:

- Integration between widgets and LLM services
- Business logic for processing AI requests and responses
- Customizable workflows for AI-assisted operations

Code Snippets¶

Content Script Snippets¶

Module Suite includes several Content Script snippets that:

- Facilitate quick implementation of common LLM-related tasks
- Provide reusable code for developers to extend AI functionalities
- Demonstrate best practices for integrating AI capabilities into Content Scripts

CARL (Content Server Artificial intelligence Resource and Liaison)¶

CARL is a feature of Module Suite, introduced with version 3.5, that implements a Content Script co-pilot based on the integration with GPT family models.

CARL Integration in Content Script Editor¶

When enabled in the Base Configuration, CARL integrates directly into the Content Script editor, offering:

- AI-assisted code completion and suggestions
- Context-aware help and documentation
- Intelligent debugging assistance

CARL Beta Status

CARL is currently in beta. As a beta feature, it may undergo changes and improvements in future releases. Users are encouraged to provide feedback to help shape its development.

```
graph TD
    A[Module Suite] --> B[OpenAI Service]
    A --> C[LLM Service]
    A --> D[LLM Integration Features]
    D --> E[Smart Pages Widget CARL]
    D --> F[Beautiful WebForm Widget CARL]
    D --> G[Content Script Service carl]
    A --> H[Content Script Snippets]
    A --> I[CARL Co-pilot Feature]
    I --> J[Content Script Editor Integration]

    B --> K[OpenAI/Azure AI]
    C --> L[Ollama]

    style A fill:#f9f,stroke:#333,stroke-width:2px
    style D fill:#bbf,stroke:#333,stroke-width:2px
    style I fill:#fbb,stroke:#333,stroke-width:2px
    style B stroke:#333,stroke-width:2px
    style C stroke:#333,stroke-width:2px
```

Integration Use Cases¶

Module Suite offers various capabilities for integrating AI-powered functionalities into your Extended ECM environment. Let's explore common use cases and how to implement them.

Chat Completion¶

Chat completion allows you to create interactive, context-aware conversations with an AI assistant. This functionality is valuable for implementing:

- Intelligent chatbots for user support
- Virtual assistants for guided ECM operations
- Interactive help systems within your ECM applications
- Natural language interfaces for complex queries or tasks

Example: Basic Chat Interaction¶

Here's a simple example of how to implement a chat completion interaction:

OpenAI ExampleCommentsOllama exampleComments

```
def systemPreamble = """You are C.A.R.L. (Content Server Artificial intelligence Resource and Liaiso
def defaultTemperature = 0.7
def model = "gpt-4"
def maxTokens = 2000

try {
    // Our services are implemented using fluent APIs and builders to simplify their usage
    // Use the auto-completion feature of the editor (CTRL+Space) to explore available configuration
    def reqBuilder = openai.newChatCompletionRequestBuilder()
        .model(model)
        .temperature(defaultTemperature)
        .maxTokens(maxTokens)
        .n(1) // Request only one completion (most common use case)

    def req = reqBuilder.build()

    // Instruct the agent about its purpose and constraints using a system message
    req.addChatMessage("system", systemPreamble)

    // Add user request or message
    req.addChatMessage("user", "Does xECM support the concept of metadata?")

    // Submit the request and synchronously wait for the response
    def result = openai.createChatCompletion(req)

    // Extract and output the content of the first (and only) choice
    out << result.choices[0].message.content
} catch (Exception e) {
    out << "An error occurred: " + e.getMessage()
}
```

This code configures the request with specific parameters such as the model to use (GPT-4), temperature setting for response randomness, and maximum token limit. The system message defines C.A.R.L.'s role, followed by a user question about xECM's metadata support. The `openai.newChatCompletionRequestBuilder()` method initializes the request, which is then configured and built. The `addChatMessage()` method is used to add both system and user messages to the conversation. Finally, the `createChatCompletion()` method sends the request to the OpenAI API and retrieves the response.

This example showcases the ease of use of the OpenAI service in Module Suite, allowing developers to quickly implement AI-powered chat functionalities within their Extended ECM environment. The service handles the complexities of API interaction, allowing developers to focus on crafting effective prompts and integrating the responses into their applications.

Use Autocompletion

Remember to use the auto-completion feature of the editor (CTRL+Space) to explore available configuration options when working with the request builder.

```
def systemPreamble = """You are C.A.R.L. (Content Server Artificial intelligence Resource and Liaiso
def defaultTemperature = 0.7
def model = "llama3"
def maxTokens = 2000
def msgID = "MyMessage"

try {
    // Use the llm service with a LangChain-based builder for Ollama
```

```
    def reqBuilder = llm.newLangChainChatCompletionRequestBuilder("ollama")
        .model(model)
        .temperature(0.0)
        .logRequestsAndResponses(true)

    // Set a custom timeout for non-OpenAI services
    reqBuilder.builder.timeout(java.time.Duration.ofSeconds(180))

    def req = reqBuilder.build()

    // Add system message to define C.A.R.L.'s role
    req.addChatMessage("system", systemPreamble)

    // Add user query
    req.addChatMessage("user", "Does xECM support the concept of metadata?")

    // Submit the request and wait for the response
    def result = llm.createChatCompletion(req)

    // Output the AI-generated response
    out << result.choices[0].message.content
} catch (Exception e) {
    out << "An error occurred: " + e.getMessage()
}
```

This example illustrates the flexibility of Module Suite's AI integration.

Key points to note:

- The llm service is used instead of a specific provider service, allowing for more generic implementations.
- The `newLangChainChatCompletionRequestBuilder` method is used with "ollama" as the provider.
- LangChain is utilized for integration with non-OpenAI services, offering additional configuration options like custom timeouts.
- The overall structure of setting up the request, adding messages, and retrieving the response remains similar to the previous example.

By using the generic llm service, you can easily switch between different AI providers or models while maintaining a consistent implementation structure. This flexibility allows you to choose the most suitable AI backend for your specific use case or requirements.

Different models different results

Experiment with different models and providers to find the best balance of performance, cost, and capabilities for your ECM AI integration needs.

Chat Completion (continued)¶

Example: Streaming Chat Completion¶

Streaming chat completion allows for real-time delivery of AI-generated responses, enhancing the responsiveness of your AI-powered applications. This is particularly useful for:

- Creating more interactive and dynamic user experiences
- Implementing typing-like effects in chatbots
- Handling long-form content generation without waiting for the entire response

Here's an example of how to implement streaming chat completion:

OpenAI ExampleComments

```
def systemPreamble = """You are C.A.R.L. (Content Server Artificial intelligence Resource and Liaiso
def defaultTemperature = 0.7
def model = "gpt-4"
def maxTokens = 2000
def msgID = "MyMessage"

try {
    def reqBuilder = openai.newChatCompletionRequestBuilder()
        .model(model)
        .temperature(defaultTemperature)
        .n(1)

    def req = reqBuilder.build()
    req.addChatMessage("system", systemPreamble)
    req.addChatMessage("user", "Does xECM support the concept of metadata?")

    def iterator = 0
    result = openai.streamChatCompletion(req, { block ->
        // This closure is invoked with streaming chunks as they become available
        def map = [
            content: block.choices?[0]?.message?.content,
            role:    block.choices?[0]?.message?.role,
            finishReason: (block.choices?[0]?.message?.content != null) ? block.choices?[0]?.finishR
        ]
        log.debug("GOT {}", map) // Log for debugging
        cache.put(msgID + "_" + iterator++, 500, map)
    })

    out << result.choices
} catch (Exception e) {
    out << "An error occurred: " + e.getMessage()
}
```

Key points about this streaming implementation:

The `streamChatCompletion` method is used instead of `createChatCompletion`. A closure is provided as the second argument to `streamChatCompletion`. This closure is called for each chunk of the response as it becomes available. The closure implements a producer-consumer pattern:

It acts as the producer, storing each chunk in the memcache service. The UI can act as the consumer, retrieving chunks from the memcache to display in real-time.

Each chunk is stored in the `memcache` with a unique key combining the msgID and an incrementing iterator. The `finishReason` is tracked to determine when the response is complete.

This approach allows for more responsive AI interactions, as the UI can start displaying the response before it's fully generated. It's particularly useful for longer responses or when you want to create a more dynamic, "typing" effect in your AI interface.

> **Use debouncing**
>
> When implementing the UI for streaming responses, consider using techniques like debouncing to balance between real-time updates and performance.

Additional considerations: Producer-Consumer Pattern¶

The streaming chat completion implementation uses a producer-consumer pattern to handle real-time data flow. Here's a sequence diagram illustrating this process:

Default Consumer Service¶

A default implementation of the consumer service, named "carl", is provided as a Content Script Service.

You can find it at: `Content Script Volume:CSTools:CARL:CSServices`

This service helps manage the consumption of streamed chat completion responses, making it easier to implement the consumer side of the producer-consumer pattern in your applications.

Sequence Diagram¶

Here's a sequence diagram illustrating the producer-consumer process, including the default "carl" service:

```
sequenceDiagram
    participant U as UI
    participant S as Script
    participant A as AI Service
    participant C as Memcache

    U->>S: Initiate chat (msgID)
    S->>A: streamChatCompletion request
    activate A
    loop For each chunk
        A-->>S: Stream chunk
        S->>C: Store chunk (msgID_i)
```

```
    S->>S: Log chunk
end
deactivate A
S-->>U: Completion notification

loop Until all chunks received
    U->>C: Request chunk (msgID_i)
    C-->>U: Return chunk
    U->>U: Update display
end
```

This diagram illustrates the following sequence:

1. The UI initiates the chat, providing a unique `msgID`.
2. The script sends a streaming chat completion request to the AI service.
3. As the AI service generates the response:
4. It streams chunks of the response back to the script.
5. The script stores each chunk in the memcache with a unique key (`msgID_i`, where `i` is an incrementing counter).
6. The script also logs each chunk for debugging purposes.
7. Once all chunks are received, the script notifies the UI that the completion is finished.
8. The UI then repeatedly requests chunks from the memcache using the `msgID` and incrementing counter.
9. As the UI receives each chunk, it updates the display, creating a real-time streaming effect.

This pattern allows for efficient handling of large responses and provides a smooth, responsive user experience. The memcache serves as a buffer between the AI service's output rate and the UI's consumption rate, ensuring that no data is lost and that the UI can process the response at its own pace.

Adapt it as needed

The actual implementation may vary depending on your specific UI needs.This diagram represents a general approach that can be adapted to various technical environments.

Benefits of Using the Default "carl" Service

When implementing streaming chat completion in your applications, consider using the provided "carl" service to streamline your development process and ensure robust handling of streamed responses.

Using the provided "carl" service offers several advantages:

1. **Simplified Implementation**: You don't need to write your own consumer logic, reducing development time and potential errors.
2. **Consistency**: The service ensures a consistent approach to consuming streamed responses across your applications.

3. **Optimization**: The service may include optimizations for efficient retrieval and assembly of chunked responses.
4. **Maintenance**: As part of Module Suite, the service will be maintained and updated, ensuring compatibility with future versions.

To use the "carl" service in your applications, you can call it from your UI code after initiating a streaming chat completion. The service will handle the retrieval and assembly of the chunked response, allowing you to focus on displaying the results to the user.

Function Calling¶

Function calling allows the AI to interact directly with Content Server, performing actions or retrieving information as needed. This powerful feature enables the AI to manipulate content and execute operations within the Extended ECM environment.

Example: Creating Folders Using AI¶

In this example, we'll demonstrate how to use function calling to create folders in Content Server based on natural language input.

OpenAI Example ChatCompletionOpenAI Example StreamChatCompletionResultComments

```
def systemPreamble = """You are C.A.R.L. (Content Server Artificial intelligence Resource and Liaiso
def defaultTemperature = 0.7
def model = "gpt-4"
def maxTokens = 2000
def msgID = "MyMessage"

try {
    def reqBuilder = openai.newChatCompletionRequestBuilder()
        .model(model)
        .temperature(defaultTemperature)
        .n(1)

    def req = reqBuilder.build()
    req.addChatMessage("system", systemPreamble)
    req.addChatMessage("user", "Create a folder for each month of the year in ${self.parent.ID}")

    // Define the function for creating a folder
    def func = openai.newChatFunctionBuilder()
        .name("createFolder")
        .description("Create a folder in the given space (identified by its ID)")
        .build()

    func.addStringParameter("folderName", "The name of the folder", true)
    func.addNumberParameter("parentID", "Parent Space Identifier", true)

    // Implement the function executor
    func.executor = { jsonArguments ->
        try {
            def slurper = new JsonSlurper()
            def args = slurper.parseText(jsonArguments)
            def newNode = docman.createFolder(docman.getNode(args.parentID as Long), args.folderName
            return "Created <a data-ampw='am-action' data-action='am_goTo' data-params='${newNode.ID
        } catch(e) {
            log.error("Unable to handle the request", e)
            return "Something went wrong"
```

```
        }
    }

    req.setFunctions([func])

    result = openai.createChatCompletion(req)
    out << result.choices[0].message
} catch (Exception e) {
    log.error("An error occurred ", e)
    out << "An error occurred: " + e.getMessage()
}
```

```
def systemPreamble = """You are C.A.R.L. (Content Server Artificial intelligence Resource and Liaiso
def defaultTemperature = 0.7
def model = "gpt-4"
def maxTokens = 2000
def msgID = "MyMessage"

try {
    def reqBuilder = openai.newChatCompletionRequestBuilder()
        .model(model)
        .temperature(defaultTemperature)
        .n(1)

    def req = reqBuilder.build()
    req.addChatMessage("system", systemPreamble)
    req.addChatMessage("user", "Create a folder for each month of the year in ${self.parent.ID}")

    // Define the function for creating a folder
    def func = openai.newChatFunctionBuilder()
        .name("createFolder")
        .description("Create a folder in the given space (identified by its ID)")
        .build()

    func.addStringParameter("folderName", "The name of the folder", true)
    func.addNumberParameter("parentID", "Parent Space Identifier", true)

    // Implement the function executor
    func.executor = { jsonArguments ->
        try {
            def slurper = new JsonSlurper()
            def args = slurper.parseText(jsonArguments)
            def newNode = docman.createFolder(docman.getNode(args.parentID as Long), args.folderName
            return "Created <a data-ampw='am-action' data-action='am_goTo' data-params='${newNode.ID
        } catch(e) {
            log.error("Unable to handle the request", e)
            return "Something went wrong"
        }
    }

    req.setFunctions([func])

    result = openai.streamChatCompletion(req, { block->  //This is an asyncrnous method, meaning tha
                                                         // won't wait for the completion to be termi
                                                         // as its second parameter a closure that wi
                                                         // chunks as they become available

        def map = [
            content:block.choices?[0]?.message?.content,
            role:   block.choices?[0]?.message?.role,
            finishReason:(block.choices?[0]?.message?.content != null)?block.choices?[0]?.finishReas
        ]
        log.debug("GOT {}", map) // Let's also print it into the log file for debugging
        cache.put(msgID+"_"+iterator++, 500, map)

    })
    out << result*.content
```

```
} catch (Exception e) {
    log.error("An error occurred ", e)
    out << "An error occurred: " + e.getMessage()
}
```

example 2

This example demonstrates several key concepts:

- **Function Definition**: We define a `createFolder` function using the `newChatFunctionBuilder()`. This function takes two parameters: `folderName` and `parentID`.
- **Function Implementation**: The executor closure contains the actual implementation of the function. It uses the Content Server API (docman) to create a new folder.
- **AI Integration**: The function is added to the chat completion request, allowing the AI to call it when necessary.
- **Natural Language Processing**: The user's request to "Create a folder for each month of the year" is interpreted by the AI, which then calls the createFolder function multiple times to fulfill the request.
- **Error Handling**: The implementation includes error handling to manage potential issues during folder creation.
- **Interactive Response**: The function returns an HTML link that allows users to navigate directly to the newly created folder.

Benefits of Function Calling

Function calling in Module Suite offers several advantages:

1. **Direct Interaction**: The AI can perform actions directly in Content Server, bridging the gap between natural language requests and system operations.
2. **Flexibility**: You can define custom functions to extend the AI's capabilities, tailoring it to your specific ECM needs.
3. **Safety**: By defining specific functions, you control what actions the AI can perform, ensuring security and preventing unintended operations.
4. **Complex Operations**: You can implement complex workflows by combining multiple function calls based on user requests.

Error handling

When implementing functions, ensure proper error handling and logging to maintain system stability and aid in troubleshooting.

Functions everywhere

Consider implementing additional functions for common ECM tasks, such as searching for documents, updating metadata, or initiating workflows. This can greatly enhance the AI's utility within your Extended ECM environment.

Document Assembly¶

Document assembly is a powerful use case that combines AI-generated content with document creation and manipulation within the Extended ECM system. This approach allows for the automatic generation of documents based on user requests, leveraging AI to create content and Module Suite's capabilities to assemble and store the document.

Example: Create a presentation letter in Word¶

OpenAI Example StreamChatCompletionResultComments

```
def systemPreamble = "You are D.O.C.S. (Document Organizing and Creation System) an AI agent tasked

data = [:]
data.company = [name:"CreativeAnswer SA", address:"Via Penate 4, 6850 Mendrisio Switzerland", descri
At CreativeAnswer, we are a united marketing agency & software house blending creative brilliance wi

We have developed a unique human-AI approach to help you create standout, cost-effective AI-driven m

This isn't your typical AI tool. We don't settle for generic AI apps, nor do we just churn out text

We bring you the most valuable approach to GenAI marketing, tailored exclusively for you. Just give

CreativeAnswer is a proud part of:
- AnswerModules Group, the award-winning Swiss tech company delivering personalized ECM software to
- Microsoft for Startups Founders Hub
"""]

data.user    = [name:"Patrick Vitali", role:"CTO"]

def defaultTemperature = 0.7
def model = "gpt-4o"
def maxTokens = 2000
def msgID = "MyMessage"
try{

    def reqBuilder  = openai.newChatCompletionRequestBuilder()
    .model(model)
    .temperature(defaultTemperature)
    .n(1)

    def req = reqBuilder.build()

    func =  llm.newOpenAIChatFunctionBuilder().name("createDocument")
            .description("Given a title, creates a new document, matching user's requests with conte

    func.parameters = []
    func.addStringParameter("title",   "The document title", true)
    func.addStringParameter("content", "The document's content. The content must be a valid  X-HTML

    func.executor = { jsonArguments ->
        try{
            def slurper = new JsonSlurper()
            def args = slurper.parseText(jsonArguments)

            if(!args.title.endsWith('.docx')){
```

```
                args.title = "${args.title}.docx"
            }

            docTemplate = docman.getNodeByPath("CreativeAnswer:Marketing:Corporate Identity:Template
            newNode = docman.getNodeByName(docTemplate.parent, args.title)

            //Load contents from a Docx file for processing
            def doc = docx.loadWordDoc(docTemplate)

            //Creates a temporary resouce
            def res = docman.getTempResource("out", "docx")

            //Updates the custom-xml databinding based on the OpenDoPE standard. Since: 2.3.0
            //Notice the combined used of multiple Content Script services: docx, docman, html, cach
            def xml = ""
            if(newNode){
                //Update xml with custom values
                xml = """<doc>
                            <content>&lt;div&gt;${html.escapeXML(html.htmlToXhtml(args.content))}&lt
                            <docID>${newNode.ID}</docID>
                        </doc>"""
                doc.updateOpenDoPEBindings(xml, true, true)
                doc.save(newNode)
            }else{
                newNode = doc.save(docTemplate.parent, args.title)
                xml = """<doc>
                            <content>&lt;div&gt;${html.escapeXML(html.htmlToXhtml(args.content))}&lt
                            <docID>${newNode.ID}</docID>
                        </doc>"""
                doc.updateOpenDoPEBindings(xml, true, true)
                doc.save(newNode)
            }
            ulrEditTemplate = "${url}?func=Edit.Edit&nodeid=${newNode.ID}&uiType=1&viewType=1&nextur

            return "The requested document has been created with title: ${args.title}. <br>You can f

        }catch(e){
            log.error("Error ",e)
            return "Something went wrong "+e.message
        }
    }

    req.setFunctions([func])

    req.addChatMessage("system", //one out: system, assistant, user
                    systemPreamble) //We instruct the agent about its purpose and constraints using
    //of the chat
    req.addChatMessage("system", "When creating documents you should consider the following context,

    //The user request
    req.addChatMessage("user", "Create a presentation letter to be sent to John Doe, ACME's CMO") //

    result = openai.streamChatCompletion(req, { block->
        def map = [
            content:block.choices?[0]?.message?.content,
            role:   block.choices?[0]?.message?.role,
            finishReason:(block.choices?[0]?.message?.content != null)?block.choices?[0]?.finishReas
        ]
        log.debug("GOT {}", map) // Let's also print it into the log file for debugging
        cache.put(msgID+"_"+iterator++, 500, map)

    })
    out << result*.content

} catch (Exception e) {
    out << "An error occurred: " + e.getMessage()
}
```

example 2

This example demonstrates how to create a presentation letter using AI-generated content and Module Suite's document manipulation features.

Key components of this implementation include:

1. **AI Content Generation**: The AI is instructed to create document content based on user requests and provided context.
2. **Document Creation Function**: A specialized function is made available to the AI for creating documents within the ECM system.
3. **Content Formatting**: The AI is guided to provide content in a specific format (X-HTML) to ensure compatibility with the document creation process.
4. **Context Provision**: Relevant data, such as company and user information, is provided to the AI to inform the content generation process.
5. **Document Template Usage**: The implementation utilizes a predefined document template as a base for new documents.
6. **Dynamic Document Update**: The system checks for existing documents with the given title, updating if found or creating new ones if not.
7. **OpenDoPE Standard**: The implementation leverages the OpenDoPE standard for XML data binding, allowing for dynamic content insertion into documents.
8. **Streaming Response**: The AI's response is streamed, enabling real-time updates and potentially faster response times for large documents.

Benefits of AI-Assisted Document Assembly

1. **Efficiency**: Automates the process of creating standard documents, saving time and reducing manual effort.
2. **Consistency**: Ensures that created documents follow a consistent structure and style.
3. **Contextual Relevance**: By providing context to the AI, the generated content can be highly relevant and personalized.
4. **Flexibility**: Can be adapted for various document types and use cases within the ECM system.
5. **Integration**: Seamlessly combines AI capabilities with existing ECM features and document templates.

Additional considerations: Implementation details¶

When implementing AI-assisted document assembly:

- Ensure that the AI is properly constrained to generate content in the required format and style.

- Regularly validate the generated content to maintain quality and consistency.
- Consider implementing error handling and logging to manage potential issues during document creation.
- Design the user interface to provide clear feedback on the document creation process and results.

Note

The specific implementation details may vary depending on your ECM environment and chosen AI service. Consult your Module Suite documentation for precise integration steps.

Tip

Consider expanding this approach to other document types, such as reports, contracts, or proposals. You can create specialized templates and AI instructions for each document type to further streamline your document creation processes.

Embedding Index Generation¶

Embedding index generation is a crucial step in creating AI-powered search and retrieval systems within your ECM environment. This use case demonstrates how Module Suite can generate embedding indexes for documents, enabling advanced semantic search capabilities.

Example: Indexing a single document¶

The provided implementation showcases the generation of an embedding index for documents in the ECM system. A key feature of this approach is its flexibility in handling different document types and chunking methods.

OpenAI CodeOllamaComments

```
try {
    // Retrieve the document node from the Enterprise Workspace
    def document = docman.getNodeByPath(docman.getEnterpriseWS(), "Documents:MS_3_7_0_Manual.pdf
    def docID = document.ID as String
    def lastVersion = document.lastVersion
    def list = null
    def sources = [:]

    // Prepare document metadata for indexing
    // We store metadata instead of raw text for efficiency:
    // 1. Document ID
    // 2. Document type (pdf or raw)
    // 3. Chunk dimension (e.g., page number or word count)
    // 4. Chunk identifier
    if (lastVersion.mimeType == "application/pdf") {
        // For PDF documents: Extract text page by page
        def content = pdf.getTextForPages(document)
        sources = content.collectEntries { entry ->
            [("${docID}_pdf_1_${entry.key}"): entry.value]
        }
```

```
        } else {
            // For non-PDF documents: Split content into word-based chunks
            def content = docman.getContentAsRawText(document)
            def chunkSize = 100 // Number of words per chunk (adjustable)
            def index = 0
            sources = content.split("\\s").toList().collate(chunkSize) *.join(" ").collectEntries {
                val -> [("${docID}_raw_${chunkSize}_${index++}"): val]
            }
        }

        // Process sources in batches of 10
        list = sources.entrySet().toList().findAll { it.value }.collate(10)
        def chunk = 0

        list.each { subEnties ->
            log.debug("Processing Chunk ${chunk} {}", subEnties *.value)
            try {
                // Generate embeddings for the current batch
                def req = llm.newOpenAIEmbeddingRequestBuilder()
                            .model("text-embedding-ada-002")
                            .input(subEnties *.value)
                            .build()
                def embeddings = llm.createEmbeddings(req).data

                // Build vectorial index using embeddings and metadata
                llm.newCSVectorialIndexManager().buildIndex(docID, embeddings, subEnties *.key)
            } catch (e) {
                // Log errors for individual chunks without stopping the entire process
                log.error("Unable to process chunk ${chunk}", e)
            }
            log.debug("End processing Chunk ${chunk++}")
        }
    } catch (e) {
        // Log any overall process errors
        log.error("An error occurred during index generation.", e)
    }
```

```
try {
    // Retrieve the document node using the provided path
    def document = docman.getNode(2031317 )
    def docID = document.ID as String
    def lastVersion = document.lastVersion
    def list = null
    def sources = [:]

    if (lastVersion.mimeType == "application/pdf") {
        // If the document is a PDF, extract text for each page and create embeddings
        def content = pdf.getTextForPages(document)
        sources = content.collectEntries { entry -> [("${docID}_pdf_1_${entry.key}"): entry.value] }
    } else {
        // If not a PDF, split the content into chunks and generate embeddings
        def content = docman.getContentAsRawText(document)
        def chunkSize = 100 // The number of words each chunk should be made of
        def index = 0
        sources = content.split("\\s").toList().collate(chunkSize) *.join(" ").collectEntries {
            val -> [("${docID}_raw_${chunkSize}_${index++}"): val]
        }

    }

    list = sources.entrySet().toList().findAll { it.value }.collate(10)
    def chunk = 0
    list.each { subEnties ->
        log.debug("Processing Chunk ${chunk} {}", subEnties *.value)
        try {
            // Create an embedding request for the chunk
```

```
            model = llm.newCSEmbeddingModelBuilder("ollama")
            .modelName("llama3")
            .logRequests(true)
            .logResponses(true)
            .build();

            def embeddings = []
            subEnties.each{
                embeddings << llm.newCSEmbedding( model.embed(it.value) )
            }

            // Build a vectorial index for the document using the embeddings
            llm.newCSVectorialIndexManager().buildIndex(docID, embeddings, subEnties*.key)
        } catch (e) {
            log.error("Unable to process chunk ${chunk}", e)
        }
        log.debug("End processing Chunk ${chunk++}")
    }
} catch (e) {
    log.error("An error occurred. ", e)
}
```

Key Features

1. **Document Type Flexibility**: The system can handle various document types, with specific handling for PDF files and a general approach for other text-based documents.
2. **Adaptive Chunking**: The chunking strategy adapts based on the document type:
3. For PDFs: Text is extracted page by page.
4. For other documents: Content is split into chunks based on a specified word count.
5. **Metadata-Rich Indexing**: Instead of storing raw text, the index stores metadata that can be used to retrieve the original text:
6. Document ID
7. Document type (PDF or raw)
8. Chunk dimensions (e.g., page number for PDFs, word count for other documents)
9. Chunk identifier
10. **Configurable Chunk Size**: For non-PDF documents, the chunk size (number of words per chunk) can be easily adjusted.
11. **Batch Processing**: Embeddings are generated and indexed in batches to manage resource usage efficiently.

Additional Considerations: Implementation Highlights¶

This implementation demonstrates several important concepts:

1. **Document Retrieval**: The system retrieves documents from the ECM using the document management API.

2. **Type-Specific Processing**: Different processing logic is applied based on the document's MIME type.
3. **Flexible Text Extraction**:
    - For PDFs, text is extracted page by page using a PDF processing service.
    - For other documents, raw text is extracted and split into word-based chunks.
4. **Metadata Generation**: Each chunk is associated with a unique identifier that includes:
    - Document ID
    - Document type (PDF or raw)
    - Chunk size or page number
    - Chunk index
5. **Embedding Generation**: The system uses an LLM service to generate embeddings for each chunk of text.
6. **Index Building**: A vectorial index is built using the generated embeddings and associated metadata.

Retrieval-Augmented Generation (RAG)¶

Retrieval-Augmented Generation (RAG) is a powerful technique that combines the strengths of large language models with the ability to access and utilize specific, up-to-date information from your Extended ECM system. This approach significantly enhances the accuracy and relevance of AI-generated responses by grounding them in your organization's actual content.

Example: Using a all the documents in a folder as a Knowledge Base¶

In this use case, we demonstrate how to implement a RAG system within the Module Suite. The system performs the following key steps:

1. Embeds the user's question using a specified embedding model.
2. Searches a pre-built set of vector indexes of your ECM content for relevant information.
3. Retrieves the actual text content of the most relevant chunks.
4. Incorporates this retrieved context into the prompt for the large language model.
5. Generates a response using the LLM, now informed by the relevant context.

> **An index for each document**
>
> This example assumes that an embedding index has been created for each document

OpenAI CodeComments

```
// Configuration parameters for the RAG system
def numberOfSnippetToConsider = 1      // Number of relevant snippets to use for context
def distanceSnippetTrashold = 0.8      // Cosine similarity threshold for relevant snippets
def embeddingModel = "text-embedding-ada-002"  // Model used for generating embeddings
def defaultTemperature = 0.8           // Temperature for LLM response generation
def model = "gpt-4"                    // LLM model to use for generating responses
def maxTokens = 2000                   // Maximum number of tokens in the LLM response
def msgID = "msgID"                    // Unique identifier for the message (used in streaming scena

// Function to retrieve relevant context based on the user's question
def getContext = { String prompt ->
    // Get all document IDs in the KB folder (assuming documents are of subtype 144)
    def indexes = docman.getNodeByPath("KB").childrenFast.findAll{it.subtype == 144}.collect{it.ID a

    // Generate embedding for the user's prompt
    def req = openai.newEmbeddingRequestBuilder().model(embeddingModel).input([prompt]).build()
    def promptEmbedding = openai.createEmbeddings(req).data

    // Search the vector indexes for similar content
    def context = llm.newCSVectorialIndexManager().getIndexesSearcher(*indexes)
                    .query(promptEmbedding[0], 100, 100)
                    .findAll{it.score >= distanceSnippetTrashold}

    def textContext = ""
    if(context) {
        // Parse the index key to extract document information
        // Format: DOCID_identifier_chunkDimension_chunkNumber
        def (docID, extMode, dimension, chunkNum) = context[0].text.split("_")

        // Retrieve the actual text content based on the extraction mode
        switch(extMode) {
            case "raw":
                // For raw text, split into words and retrieve the specific chunk
                textContext = docman.getContentAsRawText(docman.getNodeFast(docID as Long))
                                .split("\\s")
                                .toList()
                                .collate(dimension as int)
                                *.join(" ")[chunkNum as int]
                break
            case "pdf":
                // For PDFs, retrieve the specific page(s)
                textContext = pdf.getTextForPages(docman.getNodeFast(docID as Long),
                                                  chunkNum as int,
                                                  (chunkNum as int) + (dimension as int))[(chunkNum as
                break
        }
    }
    return textContext
}

// Main execution
def question = "Which is the biggest update on Module Suite 3.2.0 ?"

// Set up the chat completion request
def builder = openai.newChatCompletionRequestBuilder()
def req = builder.model(model)
                .user("u" + users.current.ID)
                .temperature(defaultTemperature)
                .n(1)
                .build()

// Add system message to define CARL's role
req.addChatMessage("system", "You're C.A.R.L. (Content Server Artificial intelligence Resource and L

// Retrieve relevant context for the question
def context = getContext(question)

// Add user message with or without context
```

```
if(context) {
    req.addChatMessage("user", """Given the following context: ${context}
Answer the user question: ${question}""")
} else {
    req.addChatMessage("user", question)
}

// Generate and output the response
result = openai.createChatCompletion(req)
out << result.choices[0].message.content
```

Key Components

1. **Embedding Generation**: The system uses the OpenAI API to generate embeddings for the user's question.
2. **Vector Index Search**: A custom `CSVectorialIndexManager` is used to search pre-built indexes of your ECM content.
3. **Context Retrieval**: Based on the search results, the system retrieves the actual text content from your ECM documents, handling different document types (e.g., raw text, PDF) appropriately.
4. **LLM Integration**: The retrieved context is incorporated into the prompt sent to the LLM (in this case, GPT-4), allowing it to generate more accurate and relevant responses.

Benefits

1. **Improved Accuracy**: By grounding the LLM's responses in your actual ECM content, the system provides more accurate and relevant answers.
2. **Up-to-date Information**: The system can access the latest information in your ECM, ensuring responses reflect the most current data.
3. **Customization**: The RAG approach allows the AI to leverage your organization's specific knowledge and terminology.
4. **Reduced Hallucination**: By providing relevant context, the likelihood of the LLM generating incorrect or fabricated information is significantly reduced.

Additional Considerations: Implementation Highlights¶

- **Index Management**: Ensure that your vector indexes are kept up-to-date as your ECM content changes.
- **Performance Optimization**: Consider caching frequently accessed content or embeddings to improve response times.
- **Privacy and Security**: Be mindful of what content is being sent to external LLM services and ensure compliance with your organization's data policies.

This implementation demonstrates a basic RAG system that can be further customized and optimized based on your specific needs and use cases within the Extended ECM environment.

Close the loop

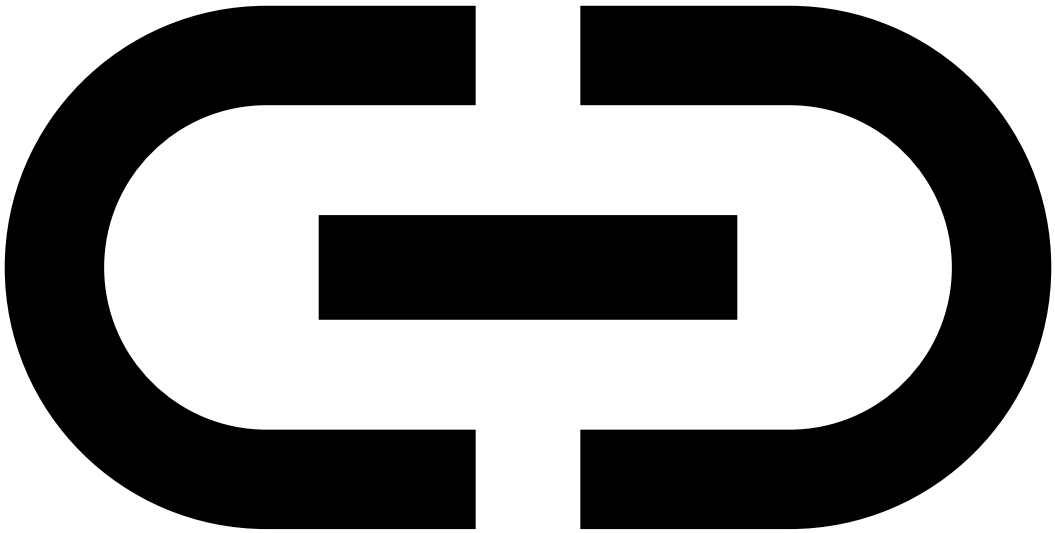
Consider implementing a feedback mechanism to continuously improve the relevance of retrieved contexts and the quality of generated responses.

Do not index just once

The effectiveness of the RAG system heavily depends on the quality and coverage of your vector indexes. Regular maintenance and updates of these indexes are crucial for optimal performance.

Configuration¶

Base Configuration



(https://developer.answermodules.com/manuals/current/administration/modulesuite/#base-configuration) parameters description, default and purpose.

CARL Service Configuration Overview¶

The following table outlines the configuration parameters for the CARL service:

Parameter	Default Value	Description
amcs.carl.enabled	false	Enable or disable CARL feature
amcs.carl.default_mode	chat	

Parameter	Default Value	Description
		Default mode for CARL (deprecated, chat is the only available mode)
amcs.carl.kb_cs_override	empty	Override for CARL co-pilot's knowledge base (KB)
amcs.carl.default_temperature	0.5D	CARL's request default temperature
amcs.carl.kb_cs_context	2	Number of CARL's KB entries to consider as context for each request
amcs.carl.kb_cs_distance	0.75D	Threshold for cosine distance among CARL's KB entries
amcs.carl.default_lang	EN	CARL's default language
amcs.carl.default_emb_model	text-embedding-ada-002	Default embedding model (used for updating the KB)
amcs.carl.kb_context_maxlen	20000	Maximum length of KB entries to be included in the request (in characters)
amcs.carl.default_maxtokens	8000	Default number of tokens to be used in completion requests
amcs.carl.auth_id	empty	Reserved for future use
amcs.carl.auth_secret	empty	The OpenAI API key for your organization
amcs.carl.auth_uri	empty	Reserved for future use
amcs.carl.api_uri	https://api.openai.com/v1/ *(https://api.openai.com/v1/)*	OpenAI API endpoint
amcs.carl.api_version	empty	Reserved for future use

Deprecated Parameter

The `amcs.carl.default_mode` parameter is considered deprecated. It exists for historical reasons when LLMs first became popular and supported text completion. Currently, almost everything is done in the form of chat completion.

Knowledge Base Override

The `amcs.carl.kb_cs_override` parameter is optional and should only be used if you want to override the knowledge base of the CARL co-pilot. The KB of CARL (the Content Script co-pilot) has been generated based on the content of the Content Script Volume, specifically the Content Script Snippets.

The carl service features an API `carl.dumpCarlKB()` that allows you to regenerate this KB so that you can save it as a document on Content Server. Under The Content Script Volume:CSTools:CARL:Utilities, there is a script named "ExportCARLKB" that implements this concept. It should be used in case you have custom snippets you want to be included in CARL's KB.

Cosine Distance Threshold

The `amcs.carl.kb_cs_distance` parameter sets the threshold for the cosine distance among CARL's KB entries. This value determines whether an entry is considered relevant and should be included in building a request context. A lower value will result in more strict matching, while a higher value will be more lenient.

Token Limit

The `amcs.carl.default_maxtokens` parameter sets the default number of tokens to be used in completion requests. Be aware that different AI models may have different maximum token limits. Ensure this value does not exceed the limit of your chosen model.

API Key Security

The `amcs.carl.auth_secret` parameter is used to store your OpenAI API key. Ensure that this value is kept secure and not exposed in any logs or public configurations. It's recommended to use secure methods for storing and managing API keys in your production environment.

API Endpoint

The `amcs.carl.api_uri` parameter is set to the default OpenAI API endpoint. If you're using a different provider or a custom endpoint, you'll need to update this value accordingly.

LLM Service Configuration Overview¶

The following table outlines the configuration parameters for the LLM service:

Parameter	Default Value	Description
amcs.llm.activeProfiles	default	Comma-separated list of active profiles
amcs.llm.provider.default	openai	The LLM API Service provider
amcs.llm.auth_id.default	empty	Used if the provider uses OAuth authentication
amcs.llm.auth_secret.default	empty	The provider's integration key (secret)

Parameter	Default Value	Description
amcs.llm.auth_uri.default	empty	The URL for generating an authorization URL (for OAuth)
amcs.llm.api_uri.default	https://api.openai.com/v1/ *(https://api.openai.com/v1/)*	The APIs endpoint
amcs.llm.model_id.default	empty	The default model to be used (deprecated)
amcs.llm.temperature.default	0.0	The default request's temperature (deprecated)
amcs.llm.net_timeout.default	3000	The default network timeout (in milliseconds)
amcs.llm.net_log_enabled.default	false	Network Request and Responses Logging
amcs.llm.index_store	empty	The storage path for embedding indexes

Active Profiles

The `amcs.llm.activeProfiles` parameter supports multiple profile configurations. You can register a new profile by adding the proper "Custom variables" to the Base Configuration. This property allows you to enable existing profiles. Profiles that are not enabled are not considered.

Supported Providers

As of the time of writing, the supported LLM API Service providers (`amcs.llm.provider.default`) are: - openai - azure - ollama

Deprecated Parameters

The following parameters are marked as deprecated: - `amcs.llm.model_id.default`: The default model to be used - `amcs.llm.temperature.default`: The default request's temperature

Consider using alternative methods to specify these values in your requests.

API Key Security

The `amcs.llm.auth_secret.default` parameter is used to store your LLM provider's API key. Ensure that this value is kept secure and not exposed in any logs or public configurations. It's recommended to use secure methods for storing and managing API keys in your production environment.

Network Logging

The `amcs.llm.net_log_enabled.default` parameter enables logging of network requests and responses. This should be set to `false` in production environments to prevent potential exposure of sensitive information.

Embedding Index Storage

The `amcs.llm.index_store` parameter specifies the storage path for embedding indexes. In most cases, this should be a shared storage location accessible to all cluster nodes (e.g., EFS) to ensure consistency across the system (e.g. `\\otadminBe01\EFS\llm_indexes\`).

OAuth Authentication

If your LLM provider uses OAuth authentication, you'll need to set the `amcs.llm.auth_id.default` and `amcs.llm.auth_uri.default` parameters accordingly. These are used for generating and managing OAuth tokens.

Defining New LLM Service Profiles¶

To create a new profile for the LLM service, you need to add custom variables to the Base Configuration. The process involves replicating the existing configuration parameters but replacing the "default" suffix with the name of your new profile. This allows you to maintain multiple configurations for different LLM providers or use cases within the same system.

For example, to create an "azure" profile for Azure AI, you would add the following custom variables:

Parameter	Value
amcs.llm.api_uri.azure	https://amai-east-us.openai.azure.com/ *(https://amai-east-us.openai.azure.com/)*
amcs.llm.auth_secret.azure	7a234aaaa666666d5a245245422cvbs23a
amcs.llm.provider.azure	azure

Similarly, to create an "ollama_prod" profile for Ollama, you would add:

Parameter	Value
amcs.llm.provider.ollama_prod	ollama
amcs.llm.api_uri.ollama_prod	http://15.201.113.243:11434 *(http://15.201.113.243:11434)*

By defining these custom variables, you create distinct profiles that can be activated using the `amcs.llm.activeProfiles` parameter. This flexibility allows you to easily switch between different LLM providers or configurations without modifying the core settings. Remember to include all necessary parameters for each profile to ensure proper functionality.

Profile Naming

Choose clear and descriptive names for your profiles to easily identify their purpose or associated provider. For instance, use names like "azure_prod", "openai_dev", or "ollama_test" to indicate both the provider and the environment.

Secure Configuration

When defining profiles, especially those containing sensitive information like API keys, ensure that your configuration files and storage are properly secured and follow your organization's security best practices.

OpenAI batch cost optimization llm

OpenAI APIs

OpenAI Batch Processing¶

Introduction¶

OpenAI Batch Processing allows you to submit multiple API requests for asynchronous processing at 50% of the cost of standard requests. All requests in a batch must be of the same type (all embeddings, all chat completions, etc.) and are organized in a JSONL (JSON Lines) file format. Requests are processed when resources are available, guaranteed within 24 hours.

How Does Batch Processing Work?¶

The batch processing workflow follows these key steps:

1. Create and upload the batch file to OpenAI
2. Create a batch job from the uploaded file
3. Monitor the batch status
4. Retrieve results once processing is complete

Examples¶

Example 1¶

◉○

Uploading a Batch FileComments

Create multiple embedding tasks and upload them as a batch file. All tasks must be of the same type:

```
// List of texts to create embeddings for
def texts = [
```

```
    "Hello world",
    "Machine learning is fascinating",
    "Natural language processing enables AI",
    "Batch processing saves costs",
    "OpenAI provides powerful AI models"
]

// Build the batch file with all tasks
// All tasks must be of the same type (all embeddings in this case)
def batchFile = openai.newBatchFileRequestBuilder("embeddings_batch")

// Create embedding requests and tasks using a closure
texts.eachWithIndex { text, i ->
    def embeddingRequest = openai.newEmbeddingRequestBuilder()
            .model("text-embedding-3-small")
            .input(text)
            .user("test-user")
            .build()

    def task = openai.newBatchTaskForEmbeddingsBuilder("embedding_${String.format('%03d', i + 1)}")
            .task(embeddingRequest)
            .build()

    batchFile.addTask(task)
}

// Build and upload the batch file
def uploadedFile = openai.uploadBatchFile(batchFile.build())
out << "File uploaded with ID: ${uploadedFile.id}"
```

This example shows how to create multiple embedding requests and tasks in a loop, then combine them into a single batch file. Each task is assigned a unique custom ID that you can use to track and identify individual requests in the batch results.

Example 2¶

Creating a Batch JobComments

After uploading a batch file, create a batch job to start processing:

```
// Use the file ID from the uploaded batch file
def fileId = "file-xxx"

// Create batch parameters for embeddings endpoint
def builder = openai.newBatchCreateParamsForEmbeddingsBuilder(fileId)

// Create the batch job
def batch = openai.createBatch(builder.build())
out << "Batch created with ID: ${batch.id}"
out << "<br>Status: ${batch.status}"
```

The batch processes asynchronously. Check status later.

Example 3¶

Retrieving Batch Status and ResultsComments

Once a batch job is created, you can check its status and retrieve results:

```
// Use the batch ID from the previous script
def batchId = "batch_xxx"

def batch = openai.retrieveBatch(batchId)
out << "Batch ID: ${batch.id}"
out << "<br>Status: ${batch.status}"
out << "<br>Created at: ${batch.createdAt}"
out << "<br>Completed at: ${batch.completedAt}"

// Check if batch is completed
if (batch.status == "completed") {

    // Retrieve batch file results
    def results = openai.retrieveBatchFileResult(batch)

    // Display batch results
    out << "<br><h3>Batch Results:</h3>"

    // Process results
    results.taskResults.each { result ->
        def embeddingResult = result.response
        out << "<br><strong>Task ID:</strong> ${result.customId}"
        out << "<br> <strong>Model:</strong> ${embeddingResult.model}"
        out << "<br> <strong>Embeddings:</strong>"

        embeddingResult.data.eachWithIndex { embedding, index ->
            def vector = embedding.embedding
            out << "<br>   [${index}] Dimension: ${vector.size()}, Sample: ${vector.take(5)}"
        }
    }
}
```

This example shows how to retrieve batch status and results. Batch status can be: `validating`, `in_progress`, `finalizing`, `completed`, `expired`, `cancelling`, or `cancelled`. Once completed, you can access the output file and process individual task results using their custom IDs.

Example 4¶

Listing All BatchesComments

List all batch jobs with pagination:

```
// List batches with a limit
def page = openai.listBatches(openai.newBatchListParamsBuilder().limit(5).build())

// Display current page results
out << "Batches on current page:"
page.data.each { batch ->
    out << "<br>  - ${batch.id}: ${batch.status}"
}

// Navigate through all pages
while (page.hasNextPage) {
    page = page.nextPage
    out << "<br>Batches on next page:"
    page.data.each { batch ->
        out << "<br>  - ${batch.id}: ${batch.status}"
```

```
    }
}
```

Use pagination to navigate through all batch jobs and monitor their statuses.

Supported Request Types¶

Batch processing supports these request types:

1. **Chat Completions**: Use `newBatchTaskForChatCompletionsBuilder()` for chat-based interactions
2. **Completions**: Use `newBatchTaskForCompletionsBuilder()` for text completion requests
3. **Embeddings**: Use `newBatchTaskForEmbeddingsBuilder()` for embedding generation
4. **Responses**: Use `newBatchTaskForResponsesBuilder()` for response API requests

Batch Status Lifecycle¶

A batch job goes through several status stages:

1. `validating`: The batch file is being validated
2. `in_progress`: The batch is being processed
3. `finalizing`: The batch is completing and results are being prepared
4. `completed`: The batch has finished successfully
5. `expired`: The batch expired before completion
6. `cancelling`: The batch is being cancelled
7. `cancelled`: The batch was cancelled

Best Practices¶

1. **Use Custom IDs**: Always assign meaningful custom IDs to your tasks to easily identify results
2. **File Organization**: Keep track of uploaded file IDs and batch IDs for result retrieval
3. **Same Task Type**: Ensure all tasks in a batch file are of the same type (all embeddings, all chat completions, etc.)

When to Use Batch Processing

Use batch processing when you have many requests to process, immediate results are not required, and you want to reduce API costs. Ideal for processing large document collections or generating embeddings for knowledge bases.

Completion Window

Batch requests are guaranteed to complete within 24 hours. Ensure your application can handle this delay.

OpenAI Evaluations (Evals)¶

Introduction¶

OpenAI Evaluations (Evals) let you test and measure AI model performance against specific criteria. Use them to validate accuracy, compare models, and ensure quality.

What are Evals Used For?¶

Evals are used to:

- **Measure Model Performance**: Evaluate how well a model performs on specific tasks or datasets
- **Compare Models**: Test different models against the same criteria to determine which performs better
- **Quality Assurance**: Ensure models meet quality standards before deployment
- **Continuous Monitoring**: Track model performance over time and detect degradation
- **A/B Testing**: Compare different model configurations or prompts

How Do Evals Work?¶

The process has two steps:

1. **Eval Creation**: Define the evaluation framework with data schema, testing criteria (graders), and metadata
2. **Eval Run Creation**: Execute the evaluation with a test data file and model specification

You can reuse the eval definition for multiple runs to test different models or datasets.

Testing Criteria (Graders)¶

Evals support different types of graders to evaluate model outputs:

- **String Check Grader**: Compares model output to a reference string using operations like `eq` (equals), `contains`, `starts_with`, or `ends_with`
- **Text Similarity Grader**: Measures semantic similarity using metrics like cosine similarity or Jaccard similarity
- **Python Grader**: Custom Python code for complex evaluation logic
- **Label Model Grader**: Uses another AI model to classify or label the output
- **Score Model Grader**: Uses an AI model to score the output on a scale

Example: Creating and Running an Eval with String Check Grader¶

This example demonstrates the complete workflow: creating an eval and then running it.

Step 1: Create the Eval¶

Create the EvalComments

```
// Define the data schema for your test cases
def itemSchema = [
    "type": "object",
    "properties": [
        "ticket_text": ["type": "string"],
        "correct_label": ["type": "string"]
    ],
    "required": ["ticket_text", "correct_label"]
]

// Create the eval with a String check grader
def evalParams = openai.newEvalCreateParamsBuilder()
    .name("Ticket Classification Evaluation")
    .dataSourceConfigOfCustom(itemSchema)
    .addStringCheckGraderTestingCriterion(
        "Match output to human label",
        '{{ sample.output_text }}',
        '{{ item.correct_label }}',
        "eq"
    )
    .build()

// Create the eval
def eval = openai.createEval(evalParams)
out << "Eval created with ID: ${eval.id}"
```

This example creates an eval that:

- Defines a data schema with `ticket_text` (the test case) and `correct_label` (the expected answer)
- Uses a String Check Grader to compare the model's output (`{{ sample.output_text }}`) with the correct label (`{{ item.correct_label }}`)
- The `eq` operation checks for exact equality

The `{{ sample.output_text }}` and `{{ item.correct_label }}` syntax uses JSONPath to reference fields from the evaluation data.

Step 2: Prepare the Eval Data File¶

Prepare the Eval Data File

Before creating an eval run, you need to prepare a JSONL (JSON Lines) file with your test data. Each line must be a valid JSON object that matches your data schema. The file should be structured as follows:

```
{ "item": { "ticket_text": "My monitor won't turn on!", "correct_label": "Hardware" } }
{ "item": { "ticket_text": "I'm in vim and I can't quit!", "correct_label": "Software" } }
{ "item": { "ticket_text": "Best restaurants in Cleveland?", "correct_label": "Other" } }
```

Each line contains an `item` object with the fields defined in your schema (`ticket_text` and `correct_label` in this example).

Step 3: Create an Eval Run¶

Create an Eval RunComments

```
// Use the eval ID from the previous script
def evalId = "eval-xxx"

// First, upload your test data file (JSONL format)
def dataFile = docman.getNodeByPath("TestData:eval_data.jsonl").content
def uploadedFile = openai.uploadFile(
    openai.newFileCreateParamsBuilder("evals", dataFile).build()
)

// Define the input messages for the model
def inputMessages = [
    openai.newEvalInputMessage(
        "developer",
        "You are an expert in categorizing IT support tickets. Given the support ticket below, categ
    ),
    openai.newEvalInputMessage(
        "user",
        '{{ item.ticket_text }}'  // Reference to the ticket_text field from your data
    )
]

// Create eval run parameters for responses
def runParams = openai.newEvalRunCreateParamsForResponsesBuilder(
    "test_run_001",
    uploadedFile.id,
    inputMessages,
    "gpt-4o"
).build()

// Create and run the eval
def evalRun = openai.createEvalRun(evalId, runParams)
out << "Eval run created with ID: ${evalRun.id}"
out << "<br>Status: ${evalRun.status}"
```

This example:

- Uploads a JSONL file containing test cases in the required format (each line is a JSON object with an `item` containing `ticket_text` and `correct_label` fields)
- Defines the prompt messages that will be sent to the model, using `{{ item.ticket_text }}` to reference the input field
- Creates an eval run that tests the `gpt-4o` model against the eval criteria
- The model's responses will be automatically graded using the String Check Grader defined in the eval, comparing `{{ sample.output_text }}` with `{{ item.correct_label }}`

You can retrieve the eval run results later to see how the model performed.

Step 4: Managing Eval Runs¶

Managing Eval RunsComments

Once you've created eval runs, you can manage them through various operations:

```
def evalId = "eval_xxx"
def runId = "evalrun_xxx"

// List all runs for an eval
def page = openai.listEvalRuns(evalId)
page.data.each { run ->
    out << "<br>Run: ${run.id} - Status: ${run.status}"
}

// Retrieve a specific eval run
def run = openai.retrieveEvalRun(
    runId,
    openai.newEvalRunRetrieveParamsBuilder(evalId).build()
)
out << "Run status: ${run.status}"

// Cancel an eval run
openai.cancelEvalRun(runId, evalId)
out << "<br>Eval run cancelled"

// Delete an eval run
openai.deleteEvalRun(runId, evalId)
out << "<br>Eval run deleted"
```

This example demonstrates how to:

- List all runs associated with an eval
- Retrieve details about a specific run, including its status and progress
- Cancel a specific eval run
- Delete an eval run

Eval runs can have different statuses such as `queued`, `in_progress`, `completed`, `cancelled`, or `failed`. Use these operations to monitor and control your evaluation runs.

Managing Evals¶

You can manage your evals through various operations:

```
// Use the eval ID from the previous script
def evalId = "eval-xxx"
// List all evals
def page = openai.listEvals(openai.newEvalListParamsBuilder().limit(10).build())
page.data.each { eval ->
    out << "<br>Eval: ${eval.name} (ID: ${eval.id})"
}

// Retrieve a specific eval
def eval = openai.retrieveEval(evalId)

// Update an eval
out << openai.updateEval(evalId, openai.newEvalUpdateParamsBuilder().name("Updated Name").build())

// Delete an eval
out << openai.deleteEval(eval.id)
```

Other Grader Types¶

In addition to String Check Grader, you can use:

- **Text Similarity Grader**: `addTextSimilarityTestingCriterion()` - Measures semantic similarity using cosine or Jaccard metrics
- **Python Grader**: `addPythonGraderTestingCriterion()` - Custom Python code for complex evaluation logic
- **Label Model Grader**: `addLabelModelTestingCriterion()` - Uses another model to classify outputs
- **Score Model Grader**: `addTestingCriterion()` - Uses a model to score outputs on a scale

Each grader type is suited for different evaluation scenarios. Choose the one that best matches your evaluation needs.

OpenAI Fine-Tuning¶

Introduction¶

Cost Warning

Fine-tuning jobs can incur significant costs, with training costs potentially reaching up to **$100 per hour** depending on the model and method used. Costs vary based on:

- The base model being fine-tuned
- The fine-tuning method (DPO, Supervised, or Reinforcement)
- The size of your training dataset
- Training duration

For the most current and detailed pricing information, refer to the OpenAI Pricing Documentation *(https://platform.openai.com/docs/pricing#fine-tuning)*. Always monitor your usage and costs when running fine-tuning jobs.

OpenAI Fine-Tuning allows you to customize models for your specific use case by training them on your own data. Fine-tuning improves model performance on specific tasks, enables you to teach the model new behaviors, and can reduce costs by allowing you to use smaller models effectively.

Fine-Tuning Methods¶

Module Suite supports three fine-tuning methods:

1. **DPO (Direct Preference Optimization)**: Optimizes models using pairwise preference data, teaching the model to favor certain outputs over others. Ideal for subjective quality improvements like tone, style, or appropriateness.
2. **Supervised Fine-Tuning**: Trains models on input-output pairs, teaching them to follow specific patterns or formats.
3. **Reinforcement Learning**: Uses reward models to guide training, suitable for complex optimization scenarios.

Example: DPO Fine-Tuning¶

This example demonstrates how to create a DPO fine-tuning job:

Step 1: Prepare DPO Training Data¶

Prepare DPO Training DataComments

DPO requires pairs of responses where one is preferred over the other. Each sample needs: - An input (the prompt/request) - A preferred output (the better response) - A non-preferred output (the less desirable response)

```
// Create a DPO fine-tuning file builder
def builderFile = openai.newDpoFineTuneFileRequestBuilder("my-dpo-training-data")

// Example: Create a chat completion request as input
def userQuestion = "How is the weather in the north pole?"
def requestBuilder = openai.newChatCompletionRequestBuilder()
    .model("gpt-4o")
    .addChatMessage("user", userQuestion)
    .build()

// Create a preferred response (more desirable answer)
def preferredBuilder = openai.newChatCompletionRequestBuilder()
    .model("gpt-4o")
    .addChatMessage("assistant", "The weather at the North Pole is extreme, with very long, dark, an
    .build()
def preferredMessage = preferredBuilder.messages[0]

// Create a non-preferred response (less desirable answer)
def nonPreferredBuilder = openai.newChatCompletionRequestBuilder()
    .model("gpt-4o")
    .addChatMessage("assistant", "The weather is hot and humid")
    .build()
def nonPreferredMessage = nonPreferredBuilder.messages[0]
```

```
// Create a DPO sample
def sample = openai.newDpoFineTuneSampleBuilder()
    .input(requestBuilder)
    .preferred(preferredMessage)
    .nonPreferred(nonPreferredMessage)
    .build()

// Add multiple samples to the file (you'll need many samples for effective training)
for (int i = 0; i < 50; i++) {
    builderFile.addSample(sample)
}

// Upload the DPO training file
def uploadedFile = openai.uploadDpoFineTuneFile(builderFile.build())
out << "File uploaded with ID: ${uploadedFile.id}"
```

Creates a DPO file with multiple samples. Each sample has an input, preferred response, and non-preferred response.

Training Data Requirements:

- The minimum number of examples you can provide for fine-tuning is 10
- We see improvements from fine-tuning on 50–100 examples, but the right number for you varies greatly and depends on the use case
- We recommend starting with 50 well-crafted demonstrations and evaluating the results

Step 2: Create the Fine-Tuning Job¶

Create the Fine-Tuning JobComments

```
// Use the uploaded file ID
def fileId = uploadedFile.id

// Create fine-tuning job parameters with DPO method
def builder = openai.newFineTuneJobCreateParamsBuilder(
    "gpt-4.1-mini-2025-04-14",  // Base model to fine-tune
    fileId                      // Training file ID
).methodDPO()                   // Use DPO fine-tuning method

// Create the fine-tuning job
def fineTuneJob = openai.createFineTuneJob(builder.build())
out << "Fine-tuning job created with ID: ${fineTuneJob.id}"
out << "<br>Status: ${fineTuneJob.status}"
```

Creates a DPO fine-tuning job with the base model and training file. Jobs process asynchronously and can take significant time depending on dataset size and model complexity.

Step 3: Managing Fine-Tuning Jobs¶

Managing Fine-Tuning JobsComments

You can monitor and manage your fine-tuning jobs:

```
// Use the finetune ID from the previous script
def fineTuneId = "ftjob-xxx"

// List all fine-tuning jobs
def page = openai.listFineTuneJobs()
page.data.each { job ->
    out << "<br>Job: ${job.id} - Status: ${job.status} - Model: ${job.model}"
}

// Navigate through pages
while (page.hasNextPage) {
    page = page.nextPage
    page.data.each { job ->
        out << "<br>Job: ${job.id} - Status: ${job.status}"
    }
}

// Retrieve a specific fine-tuning job
def job = openai.retrieveFineTuneJob(fineTuneId)
out << "Status: ${job.status}"
out << "<br>Trained tokens: ${job.trainedTokens}"

// ONLY reinformecent job can be paused and resumed
// Pause a fine-tuning job
/*def pausedJob = openai.pauseFineTuneJob(job.id)
out << "<br>Job paused: ${pausedJob.status}"*/

// Resume a paused fine-tuning job
/*def resumedJob = openai.resumeFineTuneJob(job.id)
out << "<br>Job resumed: ${resumedJob.status}"*/
```

List jobs with pagination, retrieve job details, and pause or resume jobs. Statuses: `validating_files`, `queued`, `running`, `succeeded`, `failed`, `cancelled`, or `paused`.

Other Fine-Tuning Methods¶

In addition to DPO, you can use:

- **Supervised Fine-Tuning**: Use `methodSupervised()` with `newSupervisedFineTuneFileRequestBuilder()` and `uploadSupervisedFineTuneFile()` - Trains on input-output pairs
- **Reinforcement Learning**: Use `methodReinforcement()` with reward models - Uses reinforcement learning with human feedback

Model Context Protocol OpenAI integration mcp

MCP

Integrate Model Context Protocol in your workflow¶

Introduction¶

The Model Context Protocol (MCP) is a standardized protocol that enables AI models to access external tools and resources through a secure, capability-based negotiation system. This protocol bridges the gap between AI models and external systems, allowing for dynamic tool discovery, execution, and integration.

Module Suite plays a crucial role in seamlessly incorporating MCP into your enterprise content management ecosystem. By bridging the gap between your organization's content repositories and MCP-compatible servers, Module Suite empowers you to:

1. **Extend AI capabilities**: Leverage external tools and services through MCP servers, expanding the functionality available to AI models.
2. **Standardize tool integration**: Use a consistent protocol for connecting AI models with external resources, simplifying integration complexity.
3. **Enable dynamic tool discovery**: Automatically discover and utilize tools available on MCP servers without hardcoding integrations.
4. **Enhance AI workflows**: Integrate MCP tools with OpenAI function calling, allowing AI models to automatically execute external tools based on user requests.
5. **Maintain security**: Implement secure authentication mechanisms (OAuth2 or custom) to protect access to external tools and resources.
6. **Support flexible authentication**: Choose between OAuth2 automatic authentication or custom authorization mechanisms based on your security requirements.

By integrating MCP through Module Suite, organizations can harness the power of standardized tool integration to transform their AI-powered workflows, leading to increased flexibility, reduced integration complexity, and improved overall productivity.

> **MCP Integration Considerations**
>
> While MCP offers significant benefits, it's important to consider factors such as authentication requirements, server compatibility, and network access when integrating MCP servers into your workflow. Module Suite provides the necessary tools and interfaces to address these considerations effectively.

Architecture and Networking¶

Module Suite acts as a central hub for communication between Extended ECM (xECM), MCP-compatible servers, and AI services.

Here's an overview of how the networking and communication work:

```
flowchart TD
    subgraph ECM["Extended ECM (xECM)"]
        MS[Module Suite]
    end

    MCP[MCP Server]
    OAuth[OAuth Provider]
    OpenAI[OpenAI Service]

    MS <--> |Internal APIs| ECM
    MS <--> |MCP Protocol| MCP
    MS <--> |OAuth 2.0| OAuth
    MS <--> |Function Calling| OpenAI

    style ECM fill:#f9f,stroke:#333,stroke-width:2px
    style MS fill:#bbf,stroke:#333,stroke-width:2px
    style MCP fill:#bfb,stroke:#333,stroke-width:2px
    style OAuth fill:#fbb,stroke:#333,stroke-width:2px
    style OpenAI fill:#fbb,stroke:#333,stroke-width:2px
```

Integration with xECM¶

Module Suite runs directly on xECM, providing seamless access to all xECM APIs. This tight integration allows for efficient data exchange and leveraging of xECM's content management capabilities.

MCP Server Communication¶

Module Suite implements communication channels to MCP-compatible servers, which can be: - Public internet services - VPN-accessible services - On-premises solutions

The **mcp** Content Script service includes methods to programmatically connect to MCP servers, list available tools, execute tools, and convert MCP tools to OpenAI function calling format to be used with the openai service directly.

> **MCP Protocol Support**

Module Suite features comprehensive support for the Model Context Protocol specification, enabling flexible integration with various MCP-compatible servers. This allows for standardized tool integration while maintaining a consistent interface.

Authentication Mechanisms¶

Module Suite supports two authentication modes for MCP integration:

1. **OAuth2 Authentication** (automatic): Full OAuth2 flow with automatic token management
2. **Custom Authorization** (manual): Flexible authentication using custom headers or tokens

Typical Communication Sequence¶

Below is a diagram illustrating a typical communication sequence when using Module Suite with xECM, an MCP server, and OpenAI:

```
sequenceDiagram
    participant User
    participant xECM
    participant Module Suite
    participant MCP Server
    participant OAuth Provider
    participant OpenAI

    User->>xECM: Request AI operation
    xECM->>Module Suite: Pass request
    Module Suite->>Module Suite: Check authentication
    alt Authentication required
        Module Suite->>OAuth Provider: Request authorization
        OAuth Provider-->>Module Suite: Return tokens
    end
    Module Suite->>MCP Server: List available tools
    MCP Server-->>Module Suite: Return tool definitions
    Module Suite->>OpenAI: Send request with MCP tools
    OpenAI-->>Module Suite: Request tool execution
    Module Suite->>MCP Server: Execute tool
    MCP Server-->>Module Suite: Return tool result
    Module Suite->>OpenAI: Return tool result
    OpenAI-->>Module Suite: Return final response
    Module Suite->>xECM: Process and format response
    xECM->>User: Display result
```

Components of the MCP Service Integration¶

Module Suite provides a comprehensive set of components to enable seamless integration with MCP services. These components work together to offer a robust and flexible MCP-enhanced experience within the Extended ECM environment.

Content Script Service¶

MCP Extension Package Service¶

The MCP service is a dedicated Content Script extension package service specifically designed for Model Context Protocol integrations. Key features include:

- Multi-profile support for flexible configuration
- Comprehensive implementation of MCP protocol features
- OAuth2 and custom authorization support
- Tool discovery and execution capabilities
- OpenAI function calling integration

```
graph TD
    A[Module Suite] --> B[MCP Service]
    B --> C[Tool Discovery]
    B --> D[Tool Execution]
    B --> E[OpenAI Integration]
    B --> F[Authentication]
    F --> G[OAuth2]
    F --> H[Custom Authorization]

    style A fill:#f9f,stroke:#333,stroke-width:2px
    style B fill:#bbf,stroke:#333,stroke-width:2px
```

Integration Use Cases¶

Module Suite offers various capabilities for integrating MCP-powered functionalities into your Extended ECM environment. Let's explore common use cases and how to implement them.

Tool Discovery¶

Tool discovery allows you to explore and understand the capabilities available on an MCP server. This functionality is valuable for implementing:

- Dynamic tool integration without hardcoding
- Capability-based feature negotiation
- Tool documentation and schema inspection

- Integration planning and development

Example: Listing Available Tools¶

Here's a simple example of how to discover tools available on an MCP server:

Basic ExampleComments

```
try {
    mcp.validateProfile("mcp-profile")
    def toolsResult = mcp.listTools("mcp-profile")

    toolsResult.tools.each { tool ->
        out << "<p><b>${tool.name}</b>: ${tool.description}</p>"
    }
} catch (Exception e) {
    log.error("Error listing tools: ${e.message}", e)
    out << "An error occurred: " + e.getMessage()
}
```

This code validates the MCP profile configuration, retrieves the list of available tools from the MCP server, and displays each tool's name and description.

The `listTools` method returns a result object containing all tools available on the MCP server. Each tool includes metadata such as name, description, and parameter schemas.

Use Autocompletion

Remember to use the auto-completion feature of the editor (CTRL+Space) to explore available configuration options when working with the MCP service.

Tool Execution¶

Tool execution allows the AI to interact directly with external systems through MCP servers, performing actions or retrieving information as needed. This powerful feature enables the AI to manipulate content and execute operations within external systems.

Example: Executing a Tool¶

In this example, we'll demonstrate how to execute a tool on an MCP server:

Basic Tool ExecutionComments

```
try {
    mcp.validateProfile("mcp-profile")

    def result = mcp.executeTool(
        "search_documents",
```

```
            [
                query: "contract",
                limit: 10
            ],
            "mcp-profile"
        )

        result.content.each { content ->
            if (content.type == "text") {
                out << "<p>${content.text}</p>"
            }
        }
    } catch (Exception e) {
        log.error("Error executing tool: ${e.message}", e)
        out << "An error occurred: " + e.getMessage()
    }
```

This example demonstrates several key concepts:

- **Profile Validation**: The `validateProfile` method ensures the MCP profile is properly configured and authenticated.
- **Tool Execution**: The `executeTool` method takes the tool name, parameters, and profile ID to execute the tool on the MCP server.
- **Result Processing**: The method returns a `CSMcpCallToolResult` object that contains:
  - **content**: List of content items returned by the tool (text, image, audio, resource, or resource_link)
  - **isError**: Boolean indicating if the tool execution resulted in an error
  - **structuredContent**: Optional JSON object representing the structured result of the tool call
  - **meta**: Map containing metadata for the result
- **Error Handling**: The implementation includes error handling to manage potential issues during tool execution.

> **Tool Parameter Validation**
>
> Ensure that tool parameters match the schema defined by the MCP server. Invalid parameters will result in tool execution failures.

MCP Tools with OpenAI Integration¶

MCP tools can be seamlessly integrated with OpenAI function calling, allowing AI models to automatically discover and execute external tools based on user requests. This integration enables powerful workflows where AI models can leverage external capabilities dynamically.

Example: Using MCP Tools with OpenAI¶

OpenAI Integration ExampleComments

```
def profileId = 'mcp-profile'

try {
    // Validate profile
    mcp.validateProfile(profileId)

    // Get all MCP tools as OpenAI response functions
    def tools = mcp.listToolsAsOpenAIResponseTools(profileId)

    // Build OpenAI response request with all tools
    def responseBuilder = openai.newResponseRequestBuilder()
        .model("gpt-4o")
        .instructions("You are a helpful assistant.")
        .input("Write a file and list all files in the directory")
        .toolChoiceMode("auto")

    // Add all tools as function tools
    tools.each { tool -> responseBuilder.addFunctionTool(tool) }

    def request = responseBuilder.build()
    def result = openai.createResponse(request)

    // Process response types
    def responseTypes = result.getResponseType()

    responseTypes.eachWithIndex { type, index ->
        if (type == "function_call") {
            // Handle function call - executor automatically executes MCP tool
            result.handleFunctionCall({ functionCallId, callId, status, name, arguments ->
                def selectedTool = tools.find { it.name == name }
                def functionResult = selectedTool?.executor(arguments)
                return functionResult
            }, index)
        }
    }
} catch (Exception e) {
    log.error("Error in OpenAI integration: ${e.message}", e)
    out << "An error occurred: " + e.getMessage()
}
```

This example demonstrates the powerful integration between MCP and OpenAI:

- **Tool Conversion**: The `listToolsAsOpenAIResponseTools` method converts MCP tools into OpenAI function calling format, making them available to AI models.
- **Automatic Tool Discovery**: All available MCP tools are automatically added to the OpenAI request, allowing the AI to choose which tools to use based on the user's request.
- **Automatic Execution**: When the AI decides to call a function, the executor automatically executes the corresponding MCP tool, handling the complexity of tool invocation.
- **Seamless Integration**: The integration is transparent to the AI model, which simply sees function calling capabilities without needing to know about the underlying MCP protocol.

Benefits of MCP-OpenAI Integration

This integration offers several advantages:

1. **Dynamic Tool Discovery**: AI models can automatically discover and use new tools as they become available on MCP servers.
2. **Standardized Interface**: MCP provides a standardized way to expose tools, making integration consistent across different services.
3. **Flexible Execution**: Tools can be executed automatically based on AI decisions, reducing the need for manual intervention.
4. **Extensibility**: New tools can be added to MCP servers without requiring changes to the AI integration code.

Tool Configuration

When creating a new MCP tool execution, it is possible to provide the tool configuration in the form of a Map object. The structure of this map is compatible with the JSON format MCP uses to define tool calls. This is the best approach to handle complex tool executions.

Core Concepts¶

Capability-Based Negotiation¶

The Model Context Protocol uses a capability-based negotiation system where clients and servers explicitly declare their supported features during initialization. Capabilities determine which protocol features and primitives are available during a session.

- **Servers declare capabilities** like tool support
- **Clients declare capabilities** for tool invocation and execution
- Both parties must respect declared capabilities throughout the session
- Additional capabilities can be negotiated through extensions to the protocol

The following diagram illustrates the MCP session lifecycle with tool listing and execution:

```
sequenceDiagram
    participant Client as Client
    participant Server as Server

    Client->>Server: Initialize client
    Server->>Client: Initialize session with capabilities
    Client->>Server: Respond with supported capabilities

    Client->>Server: List tools
    Server->>Client: Return available tools

    loop Tool Execution
        Client->>Server: Execute tool with parameters
```

```
        Server->>Client: Return tool result
    end

    Client->>Server: Terminate
    Server->>Client: End session
```

Each capability unlocks specific protocol features for use during the session. For example:

- Implemented server features must be advertised in the server's capabilities
- Tool invocation requires the server to declare tool capabilities
- Tool listing requires the server to declare tool support in its capabilities

This capability negotiation ensures clients and servers have a clear understanding of supported functionality while maintaining protocol extensibility.

Tools¶

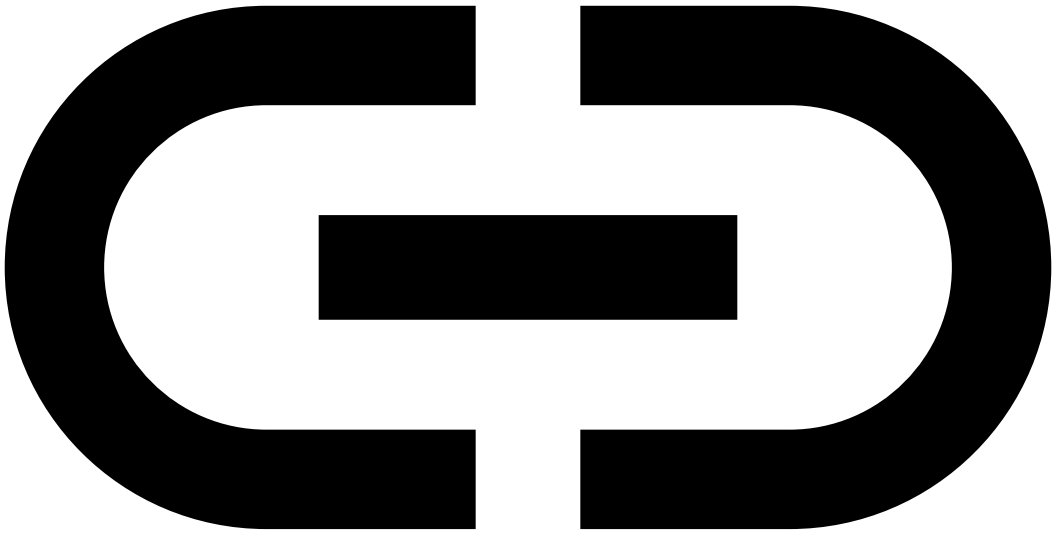
One of the core concepts when working with MCP is the "**Tool**", which represents a callable function that can be executed on the MCP server.

When defining a tool execution, you will be able to provide the tool name and parameters. The tool execution returns results that can contain text, images, resources, or other content types.

MCP tools can also be converted to OpenAI function calling format, allowing AI models to automatically execute tools through the MCP service.

Configuration¶

Base Configuration



(https://developer.answermodules.com/manuals/current/administration/modulesuite/#base-configuration) parameters description, default and purpose.

MCP Service Configuration Overview¶

The **Module Suite Extension for MCP** supports two authentication modes:

1. **OAuth2 Authentication** (automatic): Requires full OAuth configuration
2. **Custom Authorization** (manual): Only requires the MCP base URI, OAuth can be bypassed

OAuth2 Configuration¶

If you want to use OAuth2 automatic authentication, it is mandatory to set an **mcp** profile in the **Module Suite Base Configuration** with the following properties:

Property	Description	Example Value
`mcp_uri`	MCP server base URI	`https://mcp.example.com/mcp`
`client_id`	OAuth2 client ID	`your-client-id`
`client_secret`	OAuth2 client secret	`your-secret-key`
`auth_uri`	OAuth2 authorization endpoint	`https://auth.example.com/oauth/authorize`
`token_uri`	OAuth2 token endpoint	`https://auth.example.com/oauth/token`
`refresh_uri`	OAuth2 refresh token endpoint	`https://auth.example.com/oauth/refresh`
`redirect_uri`	Content Server redirect URL	`https://your-domain:8443/otcs/cs`
`scope`	OAuth2 scope	`read write or empty`
`additional_props`	Additional properties specific for MCP	`additional_key@additional_value`

OAuth2 Flow

The extension handles the OAuth authentication flow to MCP but the first time an **mcp** profile is used, it is necessary to authenticate the application on the MCP server website.

Redirect URI Configuration

The `redirect_uri` should point to your Content Server instance and will be used during the OAuth authorization flow. Ensure this URL is accessible and properly configured in your OAuth provider.

Custom Authorization Configuration¶

OAuth can be bypassed by providing an authorization closure to the `validateProfile` method. This allows authentication with any server using custom headers or authentication mechanisms. When using custom authorization, the minimum required configuration is:

Property	Description	Example Value
`mcp_uri`	MCP server base URI	`https://mcp.example.com/mcp`

All other OAuth-related properties (`client_id`, `client_secret`, `auth_uri`, etc.) are only required for OAuth automatic authentication and are not used when using custom authorization.

Security Considerations

When using custom authorization, ensure that your authentication mechanism is secure and follows your organization's security policies. API keys and tokens should be stored securely and not exposed in logs or public configurations.

Authentication¶

The extension supports two authentication modes: OAuth2 automatic authentication and custom authorization via closure.

OAuth2 Authentication¶

The MCP integration uses OAuth 2.0 for secure authentication. The following sequence diagram illustrates the complete authentication flow:

```
sequenceDiagram
    participant User as User
    participant CS as Content Server
    participant OAuth as OAuth Provider
    participant MCP as MCP Server

    User->>CS: Request MCP operation
    CS->>CS: Check for valid access token
    alt No valid access token
        CS->>CS: Check for refresh token
        alt No refresh token
            CS->>User: Redirect to authorization
            User->>OAuth: Authorize application
            OAuth->>CS: Return authorization code
            CS->>OAuth: Exchange code for tokens
            OAuth->>CS: Return access & refresh tokens
            CS->>CS: Store tokens securely
        else Refresh token exists
            CS->>OAuth: Request new access token
            OAuth->>CS: Return new access token
            CS->>CS: Store updated token
        end
        CS->>MCP: Make API request with token
        MCP->>CS: Return response
        CS->>User: Return result
    else Valid access token exists
        CS->>MCP: Make API request with token
        MCP->>CS: Return response
        CS->>User: Return result
    end
```

Token Management¶

The extension automatically handles token refresh:

- **Access tokens** are refreshed automatically when expired
- **Refresh tokens** are used to obtain new access tokens
- Tokens are stored securely in Content Server's system data
- Automatic refresh occurs transparently during API calls

Custom Authorization¶

OAuth can be bypassed by providing an authorization closure to the `validateProfile` method. This allows authentication with any server using custom headers, API keys, or other authentication mechanisms. When using custom authorization, only the `mcp_uri` configuration property is required.

EXAMPLE: Authorizing the application on the MCP endpoint¶

```
def profileId = 'mcp-profile'

try {
    mcp.validateProfile(profileId)
    out << "Successfully logged on MCP with the profile <b>${profileId}</b>"
} catch(AuthenticationException e) {
    def redirectUrl = "${url}/runcs/${self.ID}"
    def authURL = mcp.getAuthorizationUrl(redirectUrl, profileId)
    redirect authURL
}
```

The script invokes the **Module Suite Extension for MCP** validateProfile method which checks the configuration properties for the provided profile ID, checks the authorization code and, if it doesn't exists in the system, throws an AuthenticationException to indicate that the login to MCP is required. This exception is designed to be caught so that the authorization process can be initiated by redirecting the user to the OAuth authorization URL.

EXAMPLE: Using custom authorization¶

```
def profileId = 'mcp-profile'

// Define custom authorization closure
Closure authorizationClosure = { builder ->
    builder.header("OTCSTicket", "xxx-token-here")
}

// Validate profile with custom authorization (OAuth is bypassed)
mcp.validateProfile(profileId, authorizationClosure)

out << "Successfully authenticated with custom authorization"
```

The authorization closure receives a request builder that can be customized with headers, authentication tokens, or any other HTTP request modifications needed for your MCP server.

This approach allows you to authenticate with servers that use non-OAuth authentication mechanisms such as API keys, custom tokens, or Content Server tickets.

Choosing the Right Authentication Method

When selecting an authentication method for your MCP integration, consider the following factors:

- **OAuth2**: Use when the MCP server supports OAuth2 and you want automatic token management
- **Custom Authorization**: Use when you need to integrate with servers using non-standard authentication or when you want more control over the authentication process

Troubleshooting¶

Common Issues¶

Authentication Errors¶

Problem: `AuthenticationException` when calling MCP methods

Solution: Verify profile configuration and ensure user has completed OAuth flow

```
try {
    mcp.validateProfile("mcp-profile")
} catch (AuthenticationException e) {
    def authURL = mcp.getAuthorizationUrl("${url}/runcs/${self.ID}", "mcp-profile")
    redirect authURL
}
```

Configuration Errors¶

Problem: `ConfigurationException` when validating profile

Solution: Check all required properties are set and property names match exactly

```
try {
    mcp.validateProfile("mcp-profile")
} catch (ConfigurationException e) {
    log.error("Configuration error: ${e.message}")
    out << "Please check your profile configuration: ${e.message}"
}
```

Tool Execution Failures¶

Problem: Tools fail to execute or return errors

Solution: Validate tool parameters match schema and check MCP server connectivity

```
try {
    def result = mcp.executeTool("tool_name", [param: "value"], "mcp-profile")
    if (result.isError) {
        log.error("Tool execution failed")
        result.content.each { content ->
            log.error("Error: ${content.text}")
        }
    }
} catch (CSMcpServiceException e) {
    log.error("Service error: ${e.message}", e)
}
```

Additional Resources¶

- Model Context Protocol Specification *(https://modelcontextprotocol.io/)*
- MCP GitHub Repository *(https://github.com/modelcontextprotocol)*
- Module Suite Documentation *(https://developer.answermodules.com/manuals/current/)*

Extension: AdobeSign

Adobe Sign Integration Guide¶

This comprehensive guide covers the **Module Suite Extension for Adobe Sign**, providing detailed documentation for setting up and using document signing processes within your Content Server environment.

Overview¶

The **Module Suite Extension for Adobe Sign** enables seamless integration between Content Server and Adobe Sign's electronic signature platform. This extension provides a comprehensive API for:

- Creating and managing signing agreements
- Uploading documents for signature
- Managing participants and workflows
- Tracking agreement status
- Downloading signed documents
- Setting up webhooks for real-time notifications
- Managing reminders and notifications

Key Benefits¶

- **Streamlined Workflow**: Direct integration with Content Server eliminates manual document handling
- **Flexible Participant Management**: Support for multiple signature types and participant roles
- **Real-time Monitoring**: Webhook support for instant status updates
- **Comprehensive API**: Full access to Adobe Sign's REST API capabilities
- **Security**: OAuth 2.0 *(https://oauth.net/2/)* authentication with token management

System Requirements¶

Before implementing the Adobe Sign integration, ensure your system meets the following requirements:

Adobe Sign Requirements¶

- Adobe Sign Developer Account *(https://opensource.adobe.com/acrobat-sign/developer_guide/index.html#get-a-free-developer-account)*
- Adobe Sign application with appropriate permissions
- Valid Adobe Sign API credentials

Content Server Requirements¶

- Module Suite Base Configuration
- Content Script API support
- Network access to Adobe Sign endpoints

Network Requirements¶

- Outbound HTTPS access to Adobe Sign APIs
- Inbound HTTPS access for webhook callbacks (if using webhooks)
- Firewall configuration to allow Adobe Sign IP ranges

For detailed system requirements, refer to the official Adobe Sign documentation *(https://helpx.adobe.com/sign/system-requirements.html)*.

Configuration¶

Step 1: Create Adobe Sign Application¶

1. Log into your Adobe Sign Developer Console *(https://secure.eu1.adobesign.com/public/login#pageId::API_APPLICATIONS)*
2. Create a new application following the Quickstart *(https://opensource.adobe.com/acrobat-sign/developer_guide/gstarted.html)*
3. Configure the OAuth redirect URI *(https://opensource.adobe.com/acrobat-sign/developer_guide/gstarted.html#configure-the-redirect-uri-on-your-server)* to `https://your-domain/your-otcs-path/runcs/am/authflow/adobesign-profile-id`
4. Note down the following credentials:
5. Client ID
6. Client Secret
7. Authorization URI
8. Token URI
9. Refresh URI

Step 2: Configure Module Suite Profile¶

Create an Adobe Sign profile in the **Module Suite Base Configuration** with the following properties:

Property	Description	Example Value
`client_id`	Adobe Sign application client ID	`3AAABLblqZhA...`
`client_secret`	Adobe Sign application client secret	`your-secret-key`
`auth_uri`	Authorization endpoint	`https://secure.eu1.adobesign.com/public/oauth/v2`
`token_uri`	Token endpoint	`https://api.eu1.adobesign.com/oauth/v2/token`
`refresh_uri`	Refresh token endpoint	`https://api.eu1.adobesign.com/oauth/v2/refresh`
`redirect_uri`	Content Server redirect URL	`https://your-domain:8443/otcs/cs`
`scope`	Application permissions	`agreement_write+agreement_read+agreement_send`
`additional_props`	Additional properties	`api_access_point@,web_access_point@`
`web_app_client_id`	Web application client ID	`UB7E5BXCXY`

Step 3: Profile Configuration Example¶

```
// Example profile configuration
def profileConfig = [
    client_id: "3AAABLblqZhA...",
    client_secret: "your-secret-key",
    auth_uri: "https://secure.eu1.adobesign.com/public/oauth/v2",
    token_uri: "https://api.eu1.adobesign.com/oauth/v2/token",
    refresh_uri: "https://api.eu1.adobesign.com/oauth/v2/refresh",
    redirect_uri: "https://your-domain:8443/otcs/cs",
    scope: "agreement_write+agreement_read+agreement_send",
    additional_props: "api_access_point@,web_access_point@",
    web_app_client_id: "UB7E5BXCXY"
]
```

Authentication Flow¶

The Adobe Sign integration uses OAuth 2.0 for secure authentication. The following sequence diagram illustrates the complete authentication flow:

```
sequenceDiagram
    participant User as User
    participant CS as Content Server
    participant AS as Adobe Sign
```

```
    participant API as Adobe Sign API

    User->>CS: Request Adobe Sign operation
    CS->>CS: Check for valid access token
    alt No valid access token
        CS->>AS: Request valid access token using refresh token
        alt Refresh token invalid
            AS->>CS: Return authentication error
            CS->>User: Authentication error, contact the Administrator
        else Refresh token valid
            AS->>CS: Return access
            CS->>CS: Store tokens securely
            CS->>API: Make API request with token
            API->>CS: Return response
            CS->>User: Return result
        end
    else Valid access token exists
        CS->>API: Make API request with token
        API->>CS: Return response
        CS->>User: Return result
    end
```

Initial Authentication¶

The first time you use an Adobe Sign profile, you must authenticate the application:

```groovy
def profileId = 'adobesign-profile'

try {
    // Validate profile and check for existing authentication
    adobesign.validateProfile(profileId)
    out << "Successfully authenticated with Adobe Sign profile: ${profileId}"

} catch(AuthenticationException e) {
    // Generate authorization URL and redirect user
    def redirectUrl = "${url}/runcs/${self.ID}"
    def authURL = adobesign.getAuthorizationUrl(profileId, redirectUrl)

    log.info("Redirecting to Adobe Sign authorization: ${authURL}")
    redirect authURL
}
```

Token Management¶

The extension automatically handles token refresh:

- **Access tokens** expire after 1 hour
- **Refresh tokens** expire after 60 days of inactivity
- Tokens are stored securely in Content Server's system data

- Automatic refresh occurs when needed

Core Concepts¶

Agreements¶

An **Agreement** is the central concept in Adobe Sign, representing a complete signing transaction. Each agreement contains:

- **Documents**: Files to be signed
- **Participants**: People who need to sign or approve, and in what order they need to do so
- **Status**: Current state of the agreement

Agreement States¶

```
stateDiagram-v2
    [*] --> DRAFT: Create Agreement
    DRAFT --> AUTHORING: Start Editing
    AUTHORING --> IN_PROCESS: Send for Signature
    IN_PROCESS --> COMPLETED: All Signatures Complete
    IN_PROCESS --> CANCELLED: Cancel Agreement
    IN_PROCESS --> EXPIRED: Time Expired
    COMPLETED --> [*]
    CANCELLED --> [*]
    EXPIRED --> [*]

    note right of DRAFT
        Agreement created but not sent
    end note

    note right of AUTHORING
        Documents can be modified
    end note

    note right of IN_PROCESS
        Active signing workflow
    end note
```

Participant Roles¶

Role	Description	Use Case
`SIGNER`	Must sign the document	Primary signers
`APPROVER`	Must approve the document	Management approval

Role	Description	Use Case
`ACCEPTOR`	Must accept terms	Contract acceptance
`FORM_FILLER`	Must fill out forms	Data collection
`CERTIFIED_RECIPIENT`	Receives copy after signing	Audit trail
`NOTARY_SIGNER`	Notarization required	Legal documents

Document Types¶

Transient Documents¶

- Temporary documents uploaded for specific agreements
- Expire after 7 days
- Used for one-time signing processes

Library Documents¶

- Reusable document templates
- Stored permanently in Adobe Sign
- Ideal for standard contracts and forms

API Reference¶

Service Methods¶

Authentication & Configuration¶

Method	Description	Parameters
`validateProfile(profileId)`	Validates configuration and authentication	`profileId` (optional)
`getAuthorizationUrl(profileId, redirectUrl, additionalParams)`	Gets OAuth authorization URL	`profileId`, `redirectUrl`, `additionalParams` (optional)
`isClientIdValid(clientId)`	Validates client ID	`clientId`

Document Management¶

Method	Description	Parameters
`uploadDocument(document, profileId)`	Upload single document	`document` (CSDocument/CSResource/File), `profileId`
`uploadDocuments(documents, profileId)`	Upload multiple documents	`documents` (List), `profileId`

Agreement Management¶

Method	Description	Parameters
`createAndSendAgreement(documents, name, participants, message, profileId)`	Create and send agreement	`documents`, `agreementName`, `participants`, `message`, `profileId`
`createAgreementDraft(documents, name, participants, message, profileId)`	Create draft agreement	`documents`, `agreementName`, `participants`, `message`, `profileId`
`getAgreementStatus(agreementId, profileId)`	Get agreement status	`agreementId`, `profileId`
`getFullAgreementStatus(agreementId, profileId)`	Get complete agreement info	`agreementId`, `profileId`
`updateAgreementState(agreementId, state, comment, notifyOthers, profileId)`	Update agreement state	`agreementId`, `state`, `comment`, `notifyOthers`, `profileId`

Document Retrieval¶

Method	Description	Parameters
`downloadAgreement(agreementId, profileId)`	Download combined PDF	`agreementId`, `profileId`
`downloadDocument(agreementId, documentId, profileId)`	Download specific document	`agreementId`, `documentId`, `profileId`
`getAuditTrail(agreementId, profileId)`	Get audit trail PDF	`agreementId`, `profileId`

Builder Classes¶

The service provides several builder classes for constructing complex objects:

Agreement Request Builder¶

```
def agreementRequest = adobesign.newAgreementRequestBuilder()
    .withName("Contract Agreement")
    .withMessage("Please review and sign this contract")
    .withElectronicSignature()
    .inProcess()
    .withExpirationTime(new Date() + 30) // 30 days from now
    .enableDocumentVisibility()
    .build()
```

Participant Set Builder¶

```
def participantSet = adobesign.newParticipantSetInfoBuilder()
    .withRole("SIGNER")
    .withOrder(1)
    .addRecipient(users.current) // Current user
    .addRecipient([email: "user@example.com", name: "John Doe"])
    .build()
```

File Info Builder¶

```
def fileInfo = adobesign.newFileInfoBuilder()
    .withTransientDocumentId("transient-doc-id")
    .withLabel("Contract Document")
    .build()
```

Usage Examples¶

The following examples demonstrate how to use the **Module Suite Extension for Adobe Sign**:

- Basic Agreement Creation
- Advanced Agreement with Multiple Participants
- Using JSON Configuration
- Library Template Management
- Agreement Status Monitoring

Basic Agreement Creation¶

```
try {
    // 1. Retrieve document from Content Server
    def contract = docman.getDocument(1234567)

    // 2. Upload document to Adobe Sign
    def documentId = adobesign.uploadDocument(contract)

    // 3. Create participant set
    def participants = [
        adobesign.createSignerGroup([
            [email: "homer@example.com", name: "Homer J. Simpson"]
        ])
    ]

    // 4. Create and send agreement
    def agreementId = adobesign.createAndSendAgreement(
        [documentId],
        "Springfield Nuclear Power Plant - Employment Contract",
        participants,
        "Dear Homer, please sign the employment contract."
    )

    out << "Agreement created successfully with ID: ${agreementId}"

} catch (Exception e) {
    log.error("Error creating agreement: ${e.message}", e)
    out << "Error: ${e.message}"
}
```

Advanced Agreement with Multiple Participants¶

```
try {
    // Upload multiple documents
    def documents = adobesign.uploadDocuments([
        docman.getDocument(1234567),
        docman.getDocument(1234568)
    ])

    // Create complex participant workflow
    def participants = [
        // First: Legal team approval
        adobesign.createParticipantSet([
            [email: "legal@company.com", name: "Legal Team"]
        ], 1, "APPROVER"),

        // Second: Department heads
        adobesign.createParticipantSet([
            [email: "hr@company.com", name: "HR Manager"],
            [email: "finance@company.com", name: "Finance Manager"]
        ], 2, "SIGNER"),

        // Third: Final approval
        adobesign.createParticipantSet([
            [email: "ceo@company.com", name: "CEO"]
        ], 3, "SIGNER")
    ]

    // Create agreement with custom configuration
    def agreementId = adobesign.createAndSendAgreement(
        documents,
        "Multi-Party Service Agreement",
        participants,
        "Please review and sign the attached service agreement. This document requires approval from
    )

    out << "Complex agreement created: ${agreementId}"

} catch (Exception e) {
    log.error("Error creating complex agreement: ${e.message}", e)
}
```

Using JSON Configuration¶

For complex agreements, you can use JSON configuration:

```
try {
    def documents = adobesign.uploadDocuments([
        docman.getNode(2350027),
        docman.getNode(2349903)
    ])

    def agreementConfig = [
        participantSetsInfo: [
            [
                role: "SIGNER",
                memberInfos: [
                    [email: "homer@example.com", name: "Homer J. Simpson"],
                    [email: "marge@example.com", name: "Marge Simpson"]
                ],
                order: 1
            ]
```

```
        ],
        name: "Family Contract Agreement",
        signatureType: "ESIGN",
        fileInfos: documents.collect { [transientDocumentId: it] },
        state: "IN_PROCESS",
        message: "Family contract requiring both signatures",
        ccs: [
            [email: "lisa@example.com"],
            [email: "bart@example.com"]
        ],
        locale: 'en_US',
        expirationTime: new Date() + 14 // 14 days from now
    ]

    def agreementId = adobesign.sendAgreement(agreementConfig)
    out << "JSON-configured agreement created: ${agreementId}"

} catch (Exception e) {
    log.error("Error creating JSON agreement: ${e.message}", e)
}
```

Library Template Management¶

```
try {
    // Upload documents for template
    def uploadedDocs = adobesign.uploadDocuments([
        docman.getDocument(1234567),
        docman.getDocument(1234568)
    ])

    // Create library files
    def libraryFiles = uploadedDocs.collect { docId ->
        adobesign.newLibraryFileBuilder()
            .withTransientDocumentId(docId)
            .build()
    }

    // Create template
    def template = adobesign.newTemplateRequestBuilder()
        .withName("Standard Contract Template")
        .shareWithUser()
        .withTemplateTypeDocument()
        .withFileInfos(libraryFiles)
        .build()

    def templateId = adobesign.createLibraryTemplate(template)
    out << "Library template created: ${templateId}"

} catch (Exception e) {
    log.error("Error creating library template: ${e.message}", e)
}
```

Agreement Status Monitoring¶

```
try {
    def agreementId = "your-agreement-id"

    // Get basic status
    def status = adobesign.getAgreementStatus(agreementId)
    out << "<p>Agreement Status: ${status}</p>"
```

```
    // Get detailed status
    def fullStatus = adobesign.getFullAgreementStatus(agreementId)
    out << "<p>Detailed Status: ${fullStatus}</p>"

    // Check if signed

    if (status == "SIGNED") {
        // Download signed documents
        def responseBody = adobesign.downloadAgreement(agreementId)
        if (responseBody) {
            CSResource signedPdf = docman.getTempResource("signed-contract.pdf", ".pdf")
            signedPdf.content.withOutputStream { os ->
                os << responseBody.byteStream()
            }
        }
    }

} catch (Exception e) {
    log.error("Error monitoring agreement: ${e.message}", e)
}
```

Advanced Features¶

Reminder Management¶

Basic Reminder¶

```
try {
    def agreementId = "your-agreement-id"

    // Send immediate reminder to next signers
    def reminderId = adobesign.sendImmediateReminder(
        agreementId,
        "Please sign the contract as soon as possible."
    )

    out << "Reminder sent: ${reminderId}"

} catch (Exception e) {
    log.error("Error sending reminder: ${e.message}", e)
}
```

Scheduled Reminders¶

```
try {
    def agreementId = "your-agreement-id"
    //Retrieve the IDs of the next signers in the flow
    def participantIds = adobesign.getNextSignersIds(agreementId)

    // Create reminder with custom schedule
    def reminderId = adobesign.sendReminder(
        agreementId,
        participantIds,
        "Friendly reminder to complete your signature",
        24, // First reminder after 24 hours
        "AGREEMENT_AVAILABILITY", // Start counting from when agreement becomes available
        "DAILY_UNTIL_SIGNED" // Send daily until signed
```

```
    )

    out << "Scheduled reminder created: ${reminderId}"

} catch (Exception e) {
    log.error("Error creating scheduled reminder: ${e.message}", e)
}
```

Deliverable Access¶

```
try {
    def agreementId = "your-agreement-id"

    // Create participant access info
    def participantAccessInfos = [
        adobesign.newParticipantAccessInfoBuilder()
            .withParticipantId("participant-id")
            .withEmail("user@example.com")
            .build()
    ]

    // Generate access URLs
    def accessResponse = adobesign.createDeliverableAccessURL(
        agreementId,
        participantAccessInfos
    )

    out << "Access URLs generated: ${accessResponse}"

} catch (Exception e) {
    log.error("Error creating deliverable access: ${e.message}", e)
}
```

Business Workspace Integration¶

The Adobe Sign integration can be seamlessly integrated with Content Server's Business Workspace feature, allowing for structured document signing workflows with category-based metadata management.

Integration Workflow¶

The following diagram illustrates the complete Business Workspace integration workflow:

```
flowchart TD
    A[Create Business Workspace] --> B[Configure Agreement Category]
    B --> C[Add Documents with Document Category]
    C --> D[Execute Agreement Creation Script]
    D --> E{Agreement Already Exists?}
    E -->|Yes| F[Display Current Status]
    E -->|No| G[Extract Agreement Details]
    G --> H[Upload Documents to Adobe Sign]
    H --> I[Create Participant Sets]
```

```
I --> J[Send Agreement for Signature]
J --> K[Update Business Workspace Status]
K --> L[Monitor Agreement Status]
L --> M{Status Changed?}
M -->|No| N[Continue Monitoring]
M -->|Yes| O[Update Local Status]
O --> P{Status = SIGNED?}
P -->|No| N
P -->|Yes| Q[Download Signed Documents]
Q --> R[Add as New Versions]
R --> S[Complete Workflow]

style A fill:#e1f5fe
style S fill:#c8e6c9
style F fill:#fff3e0
style Q fill:#f3e5f5
```

Category Structure¶

Create the following category structure for managing agreements:

Agreement Category¶

- **Agreement Name** (Text, 254 chars): Name shown to participants
- **Message** (Text, 254 chars): Message sent to participants
- **Participants** (Set):
- **Role** (Text, 254 chars): SIGNER, APPROVER, etc.
- **Order** (Integer): Workflow order
- **User** (User Field): Content Server user
- **Full Name** (Text, 254 chars): Participant name
- **Email** (Text, 254 chars): Participant email
- **Signature State** (Set):
- **Agreement ID** (Text, 254 chars): Adobe Sign agreement ID
- **Status** (Text, 254 chars): Current status

Document Category¶

- **Transient Document ID** (Text, MultiLine): Adobe Sign document ID
- **AM Unique ID** (Text, 254 chars): Unique identifier for linking

System Architecture¶

The following diagram shows the system architecture and data flow for Business Workspace integration:

```
graph TB
    subgraph "Content Server"
        BW[Business Workspace]
        AC[Agreement Category]
        DC[Document Category]
        CS[Content Server Documents]
        AS[Adobe Sign Service]
    end

    subgraph "Adobe Sign Cloud"
        ASAPI[Adobe Sign API]
        AD[Agreement Data]
        TD[Transient Documents]
        SD[Signed Documents]
    end

    subgraph "Monitoring"
        POLL[Polling Script]
        WH[Webhook Handler]
    end

    BW --> AC
    BW --> DC
    BW --> CS
    AS --> ASAPI
    ASAPI --> AD
    ASAPI --> TD
    ASAPI --> SD

    POLL --> AS
    WH --> AS

    ASAPI -.->|Status Updates| POLL
    ASAPI -.->|Webhook Events| WH

    style BW fill:#e3f2fd
    style AS fill:#f3e5f5
    style ASAPI fill:#fff3e0
    style POLL fill:#e8f5e8
    style WH fill:#e8f5e8
```

Business Workspace Setup¶

1. Create a category folder **Adobe Sign** containing the **Agreement** and **Document** categories

2. Create a Business Workspace structure that uses these categories
3. When creating a Business Workspace, configure the **Agreement category data**
4. Assign the **Document category** to all documents within the Business Workspace

Complete Business Workspace Script¶

The following script demonstrates a complete Business Workspace integration that handles agreement creation, status monitoring, and document synchronization:

```
/*
 * Complete Business Workspace integration script
 * Handles agreement creation, status monitoring, and document synchronization
 */

def bwID = params.bwID

try {
    if (!bwID) {
        throw new ExecutionFaultException("Business Workspace ID is required")
    }

    // Get Business Workspace
    CSNode bw = docman.getNode(bwID)
    if (!bw) {
        throw new ExecutionFaultException("Business Workspace not found: ${bwID}")
    }

    def bwNameId = "Business Workspace ${bw.name} (${bw.ID})"

    // Check if already processed
    def agreementCat = bw."Agreement".asType(Map)
    def agreementID = agreementCat["Signature State"]["Agreement ID"].flatten()[0]
    def agreementStatus = agreementCat["Signature State"]["Status"].flatten()[0]

    if (agreementID) {
        out << "${bwNameId} already sent for signature.<br>ID: <b>${agreementID}</b>, Status: <b>${a
        return
    }

    // Extract agreement details
    def agreementName = agreementCat["Agreement Name"][0] ?: bw.name
    def message = agreementCat["Message"][0] ?: "Please review and sign the attached documents"
    def participants = agreementCat["Participants"]

    if (!participants.size()) {
        throw new ExecutionFaultException("${bwNameId} has no participants configured")
    }

    // Get documents from Business Workspace
    def bwDocuments = bw.getChildrenFast()
    if (!bwDocuments) {
        throw new ExecutionFaultException("${bwNameId} has no documents to sign")
    }

    // Upload documents and create file info objects
    def documentsNodes = bwDocuments.collect { it.getOriginalNode() }
    def transientDocumentsIds = adobesign.uploadDocuments(documentsNodes)
    def transientDocuments = []

    transientDocumentsIds.eachWithIndex { docId, index ->
        def amUniqueId = UUID.randomUUID().toString()

        // Update document category
```

```
        bwDocuments[index]."Document"."Transient Document ID" = docId
        bwDocuments[index]."Document"."AM Unique ID" = amUniqueId
        bwDocuments[index].update()

        // Create file info for agreement
        transientDocuments.add(
            adobesign.newFileInfoBuilder()
                .withTransientDocumentId(docId)
                .withLabel(amUniqueId)
                .build()
        )
    }

    // Build participant sets from category data
    def participantSets = []
    def rolesKeys = participants*."Role".unique()

    rolesKeys.each { roleKey ->
        def orders = agreementCat["Participants"]
            .findAll { it."Role" == roleKey }
            .sort { it."Order" }
            *."Order".unique()

        orders.each { order ->
            def orderedParticipants = agreementCat["Participants"]
                .findAll { it."Role"[0] == roleKey[0] && it."Order"[0] == order[0] }

            def memberInfos = orderedParticipants.collect {
                if (it."User"[0]) {
                    users.getMemberById(it."User"[0])
                } else {
                    [name: it."Full Name"[0], email: it."Email"[0]]
                }
            }

            participantSets.add(
                adobesign.newParticipantSetInfoBuilder()
                    .withRole(roleKey[0].toUpperCase())
                    .withOrder(order[0] as Integer)
                    .withParticipantsFromList(memberInfos)
                    .build()
            )
        }
    }

    // Create and send agreement
    agreementID = adobesign.createAndSendAgreementWithTransientDocs(
        transientDocuments,
        agreementName,
        participantSets,
        message
    )

    if (agreementID) {
        // Update Business Workspace with agreement info
        bw."Agreement"."Signature State"."Agreement ID" = agreementID
        bw."Agreement"."Signature State"."Status" = "IN_PROCESS"
        bw.update()

        out << "Agreement created successfully for ${bwNameId}<br>Agreement ID: <b>${agreementID}</b
        log.info("Agreement created for Business Workspace ${bwNameId}: ${agreementID}")
    }

} catch (Exception e) {
    log.error("Error processing Business Workspace ${bwID}: ${e.message}", e)
    out << "Error: ${e.message}"
}
```

The Agreement is now created and the documents are being sent to signature. The next step is to check the status of the Agreement and, when SIGNED, to download the signed version of each document.

Status Monitoring and Synchronization¶

An important action to be performed when a signing workflow is concluded is to retrieve the signed documents and synchronize them back on your Content Server system. Module Suite Extension for Adobe Sign doesn't provide an automation, it's up to you to handle these operations in your Content Server system.

Monitoring Approaches¶

The following diagram compares the two monitoring approaches:

```
graph LR
    subgraph "Polling Approach"
        P1[Scheduled Script] --> P2[Check All Agreements]
        P2 --> P3[Query Adobe Sign API]
        P3 --> P4[Update Local Status]
        P4 --> P5[Download if Signed]
        P5 --> P6[Wait for Next Schedule]
        P6 --> P1
    end

    subgraph "Webhook Approach"
        W1[Agreement Status Change] --> W2[Adobe Sign Webhook]
        W2 --> W3[Content Server Callback]
        W3 --> W4[Process Event]
        W4 --> W5[Update Local Status]
        W5 --> W6[Download if Signed]
    end

    style P1 fill:#e3f2fd
    style W1 fill:#f3e5f5
    style P5 fill:#c8e6c9
    style W6 fill:#c8e6c9
```

To retrieve the signed documents there are two different approaches:

- Poll Adobe Sign for the agreement status and update the local instance when a change is detected
- Subscribe to Adobe Sign push notifications for the Agreement status changes (webhook pattern)

Polling-Based Monitoring¶

One way to retrieve the status updates for the Agreement is to poll Adobe Sign to detect changes. It can be implemented by using the `getAgreementStatus(...)` API on the **adobesign** service.

The following code example can be scheduled to periodically check and update all the Adobe Sign agreements in your system (in our scenario, the Business Workspaces root directory).

> **Correct API Usage**
>
> Adobe Sign monitors that the usage of the API is compliant with certain guidelines. Specifically, certain APIs cannot be invoked with a frequency that goes over a certain threshold. When scheduling polling scripts, make sure that the scheduling frequency complies with the Adobe Sign guidelines.

```
/*
 * Scheduled script for monitoring agreement status
 * Run this script periodically to check for status changes
 */

import org.apache.commons.io.FilenameUtils

// The path to the Adobe Sign Workspace, modify it to match your setup
def adobeSignWorkspacePath = "Workspaces:Adobe Sign Workspace"

// Retrieves the information about the category in order to be able to retrieve a specific document
def adobeSignDocumentCategory                = docman.getCategory("Adobe Sign:Document")
def adobeSignDocumentCategoryDefinitionID    = adobeSignDocumentCategory.definitionID
def adobeSignAMUniqueIdAttributeID           = adobeSignDocumentCategory.getAttributeID('AM Unique I

// The method which peforms query to retrieve the specific document via the "AM Unique ID" category
def getDocumentByLabel = { amUniqueId ->
    try{
        // Define SQL code
        def sqlCode = """
SELECT
        D.DataID "DATAID"
FROM    DTree D INNER JOIN LLAttrData LLA
    ON D.DataID = LLA.ID AND D.VersionNum = LLA.VerNum
WHERE
    LLA.AttrID          = ${adobeSignAMUniqueIdAttributeID}
    AND LLA.DefID       = ${adobeSignDocumentCategoryDefinitionID}
    AND LLA.ValStr      = '${amUniqueId}'
"""
        // Set cursor and transaction enabled flags
        Boolean cursorEnabled = true
        Boolean transactionEnabled = true
        // Set record limit
        Integer recordLimit = 1

        // Run the SQL query
        def result = docman.runSQL(sqlCode,  cursorEnabled,  transactionEnabled,  recordLimit).rows

        return result[0].DATAID
    } catch(e) {
        log.error("Unable to retrieve the Document with the AM Unique ID  ${amUniqueId}", e)
    }
}

try {
    def workspacesNode = docman.getNodeByPath(adobeSignWorkspacePath)
```

```
    def businessWorkspaces = docman.getChildrenFast(workspacesNode).findAll { it.subtype == 848 }

    businessWorkspaces.each { bw ->
        def agreementID = bw."Agreement"."Signature State"."Agreement ID"[0]

        if (agreementID) {
            try {
                def agreementStatus = adobesign.getAgreementStatus(agreementID)
                def currentStatus = bw."Agreement"."Signature State"."Status"[0]

                if (agreementStatus.status != currentStatus) {
                    // Status changed - update local system
                    bw."Agreement"."Signature State"."Status" = agreementStatus.status
                    bw.update()

                    log.info("Agreement ${agreementID} status updated: ${currentStatus} -> ${agreeme

                    // Handle completed agreements
                    if (agreementStatus.status == "SIGNED") {
                        processSignedAgreement(agreementID, bw)
                    }
                }

            } catch (Exception e) {
                log.error("Error checking status for agreement ${agreementID}: ${e.message}", e)
            }
        }
    }

} catch (Exception e) {
    log.error("Error in agreement monitoring: ${e.message}", e)
}

def processSignedAgreement(agreementID, businessWorkspace) {
    try {
        // Get agreement documents
        def documents = adobesign.getAgreementDocuments(agreementID)

        if (documents?.documents) {
            documents.documents.each { doc ->
                // Find corresponding Content Server document
                def bwDocument = getDocumentByLabel(doc.label)

                if (bwDocument) {
                    // Download signed document
                    def responseBody = adobesign.downloadDocument(agreementID, doc.id)

                    if (responseBody) {
                        // Save signed version
                        CSResource signedPdf = docman.getTempResource(
                            FilenameUtils.getBaseName(bwDocument.name),
                            ".pdf"
                        )

                        signedPdf.content.withOutputStream { os ->
                            os << responseBody.byteStream()
                        }

                        // Add as new version
                        if (bwDocument.getOriginalNode()) {
                            bwDocument.getOriginalNode().addVersion(signedPdf.content)
                        } else {
                            bwDocument.addVersion(signedPdf.content)
                        }

                        log.info("Added signed version for document: ${bwDocument.name}")
                    }
                }
            }
```

```
        }

    } catch (Exception e) {
        log.error("Error processing signed agreement ${agreementID}: ${e.message}", e)
    }
}
```

Webhook-Based Monitoring¶

Another way to retrieve the Agreement status and the signed Documents is to subscribe to the Adobe Sign push notification. This solution doesn't suffer from the limitations of the polling approach such as the number of requests to Adobe Sign because it will be Adobe Sign itself that pushes the notification when necessary.

To have a better understanding of this pattern, take a look to the Adobe Sign Webhook overview *(https://helpx.adobe.com/sign/developer/webhook/overview.html)*.

To prevent any malicious attack, explicitly allow the Adobe Sign IP ranges for webhooks in your Application Server/Firewall. Take a look to the System Requirements for Adobe Sign *(https://helpx.adobe.com/sign/system-requirements.html)*.

In order to handle payloads from the Adobe Sign webhooks, the *Script Console Extension for Adobe Sign* must be installed.

How to create an Adobe Sign webhook¶

There are two ways to handle a webhook in Adobe Sign: - through the **Adobe Sign Administration Panel** - using the **Module Suite Extension for Adobe Sign** APIs

Creating a webhook using the Module Suite Extension for Adobe Sign¶

```
try {
    def webhookId = adobesign.createAgreementWebhook(
        "WEBHOOK_FROM_SCRIPT",
        "https://your_address:8443/csconsole/ext/adobesign/adobeSign.cs",
        ["AGREEMENT_CREATED", "AGREEMENT_ACTION_COMPLETED", "AGREEMENT_WORKFLOW_COMPLETED"],
        true, // Include detailed info
        true, // Include participants info
        true, // Include documents info
        true  // Include signed documents
    )

    out << "Webhook created successfully: ${webhookId}"

} catch (Exception e) {
    log.error("Error while creating the webhook on Adobe Sign: ${e.message}", e)
}
```

Deleting a webhook using the Module Suite Extension for Adobe Sign¶

```
try {
    adobesign.deleteWebhook(webhookId)
    out << "Webhook deleted successfully"

} catch (Exception e) {
    log.error("Error while deleting the webhook on Adobe Sign: ${e.message}", e)
}
```

How to handle webhook payloads¶

After installing the extension for the Script Console it is necessary to create an asyncronous callback script for the directory **Adobe Sign Sync Inbox**. This directory is the receptacle of the payloads from the Adobe Sign webhook events *(https://helpx.adobe.com/sign/fedramp/api/webhook-event-payloads.html)*.

Warning

The **Module Suite Script Console Extension for Adobe Sign** only provides an automation to handle the webhook payloads in your Content Server system under the directory Adobe Sign Sync Inbox, if configured. It's up to you to define an asyncronous callback script to convert the payload in your system.

The following script is an example of callback that saves the signed document in the Business Workspace structure.

```
import org.apache.commons.io.FilenameUtils;

// Child node create is triggered when you add to a container
// a node that was not already on Content Server

log.debug( "Executing Adobe Sign synchronization callback. Adobe Sign data doc: '{}', parent folder:

def adobeSignAgreementCategory
def adobeSignAgreementCategoryDefinitionID
def adobeSignAgreementIDAttributeID
def adobeSignDocumentCategory
def adobeSignDocumentCategoryDefinitionID
def adobeSignAMUniqueIdAttributeID

try {

    // Retrieves the information about the category in order to be able to retrieve a specific works
    adobeSignAgreementCategory                = docman.getCategory("Adobe Sign:Agreement")
    adobeSignAgreementCategoryDefinitionID    = adobeSignAgreementCategory.definitionID
    adobeSignAgreementIDAttributeID           = adobeSignAgreementCategory.getAttributeID('Agreement

    // Retrieves the information about the category in order to be able to retrieve a specific docum
    adobeSignDocumentCategory                 = docman.getCategory("Adobe Sign:Document")
    adobeSignDocumentCategoryDefinitionID     = adobeSignDocumentCategory.definitionID
    adobeSignAMUniqueIdAttributeID            = adobeSignDocumentCategory.getAttributeID('AM Unique I

} catch( Exception e ) {

    log.error("Error retrieving categories for Adobe Sign synchronization callback:" , e)

}
```

```
// The method which peforms query to retrieve the specific document via category attribute
def getNodeByCatAttrValue = { defID, attrID, valStr ->
    try{
        // Define SQL code
        def sqlCode = """
SELECT
        D.DataID "DATAID"
FROM    DTree D INNER JOIN LLAttrData LLA
    ON D.DataID = LLA.ID AND D.VersionNum = LLA.VerNum
WHERE
    LLA.DefID       = ${defID}
    AND LLA.AttrID  = ${attrID}
    AND LLA.ValStr  = '${valStr}'
"""
        // Set cursor and transaction enabled flags
        Boolean cursorEnabled = true
        Boolean transactionEnabled = true
        // Set record limit
        Integer recordLimit = 1

        // Run the SQL query
        def result = docman.runSQL(sqlCode,  cursorEnabled,  transactionEnabled,  recordLimit).rows

        return result[0]?.DATAID
    } catch(e) {
        log.error("Unable to retrieve the Document with the AM Unique ID  ${valStr}", e)
    }
}

try{
    def newDocument = asCSNode(newNodeID)

    if( newDocument.mimeType == "application/json" ){

        def jsonSlurper = new JsonSlurper()
        def jsonResponse = jsonSlurper.parseText(newDocument.content.content.text)
        def agreement = jsonResponse.agreement

        if (agreement){

            def agreementID = agreement.id
            // Retrieves the Workspace linked to the agreement via the Agreement ID category attribu
            def bwID = getNodeByCatAttrValue(adobeSignAgreementCategoryDefinitionID, adobeSignAgreem
            if (bwID){
                def bWorkspace = docman.getNodeFast(bwID)

                if (bWorkspace){

                    bWorkspace."Agreement"."Status" = agreement.status
                    bWorkspace.update()

                    // If the workflow as been completed and the agreement is SIGNED, download and s
                    if (agreement.status == "SIGNED"){
                        log.debug("#### Process Adobe Sign Webhook File - agreement signed: {}", agr

                        agreement.documentsInfo.documents.each{

                            // Retrieves the Content Server document linked to the agreement documen
                            def bwDocumentID = getNodeByCatAttrValue(adobeSignDocumentCategoryDefini

                            if (bwDocumentID) {
                                def bwDocument = docman.getNodeFast(bwDocumentID)

                                if (bwDocument){

                                    // Download the signed document
                                    def responseBody = adobesign.downloadDocument(agreementID, it.id
```

```
                                    if (responseBody){

                                        CSResource pdfFile = docman.getTempResource(FilenameUtils.ge

                                        pdfFile.content.withOutputStream { os ->
                                            os << responseBody.byteStream()
                                        }

                                        // Add the signed version to the original node, if exists
                                        if (bwDocument instanceof CSGeneration) {

                                            bwDocument.getOriginalNode().addVersion(pdfFile.content)

                                        } else {

                                            bwDocument.addVersion(pdfFile.content)
                                        }

                                        log.error("Added signed version for the document ${bwDocumen
                                    }
                                }
                            }
                        }
                    }
                }
            }
        }

    }

} catch( Exception e ) {

    log.error("Error synchronizing Adobe Sign data. Data file ID: {}", newNodeID, e)

}
```

Troubleshooting¶

Common Issues¶

Authentication Errors¶

Problem: `AuthenticationException` when calling Adobe Sign methods

Solution: 1. Verify profile configuration 2. Check if user has completed OAuth flow 3. Ensure tokens are not expired

```
try {
    adobesign.validateProfile("your-profile-id")
} catch (AuthenticationException e) {
    // Redirect to authorization
    def authURL = adobesign.getAuthorizationUrl("your-profile-id", redirectUrl)
    redirect authURL
}
```

Document Upload Failures¶

Problem: Documents fail to upload to Adobe Sign

Solution: 1. Check document format (PDF recommended) 2. Verify document size limits 3. Ensure network connectivity

```
try {
    def documentId = adobesign.uploadDocument(document)
} catch (CSAdobeSignDocumentUploadException e) {
    log.error("Document upload failed: ${e.message}")
    // Handle specific upload errors
}
```

Agreement Creation Failures¶

Problem: Agreements fail to create or send

Solution: 1. Validate participant information 2. Check document requirements 3. Verify agreement configuration

```
try {
    def agreementId = adobesign.createAndSendAgreement(documents, name, participants, message)
} catch (ExecutionFaultException e) {
    log.error("Agreement creation failed: ${e.message}")
    // Check specific error details
}
```

Debugging Tips¶

1. **Enable Detailed Logging**:

```
log.setLevel(Level.DEBUG)
```

2. **Validate Configuration**:

```
def isValid = adobesign.isClientIdValid("your-client-id")
```

3. **Check Token Status**:

```
def token = adobesign.getAccessTokenFromStore("profile-id")
if (token) {
    log.info("Token expires: ${token.expireDate}")
}
```

4. **Test API Connectivity**:

```
try {
    def baseUris = adobesign.getBaseUris("profile-id")
    log.info("Connected to Adobe Sign: ${baseUris}")
} catch (Exception e) {
    log.error("Connection failed: ${e.message}")
}
```

Performance Considerations¶

1. **Batch Operations**: Use `uploadDocuments()` for multiple files
2. **Async Processing**: Use webhooks instead of polling when possible
3. **Error Handling**: Implement proper retry logic for transient failures

Security Best Practices¶

1. **Access Control**: Limit Adobe Sign profile access to authorized users
2. **Network Security**: Use HTTPS for all communications
3. **Audit Trail**: Log all Adobe Sign operations for compliance

Additional Resources¶

- Adobe Sign REST API Documentation *(https://secure.eu1.adobesign.com/public/docs/restapi/v6)*
- Adobe Sign Developer Guide *(https://opensource.adobe.com/acrobat-sign/developer_guide/)*
- Module Suite Documentation *(https://developer.answermodules.com/manuals/current/)*
- Content Script API Reference

This guide provides comprehensive coverage of the Module Suite Extension for Adobe Sign. For additional support or advanced use cases, please refer to the official documentation or contact your system administrator.

Beautiful WebForms

Getting Started with Beautiful WebForms¶

This guide provides a quick introduction to **Module Suite Beautiful WebForms** and helps you get started with creating custom web forms and form-based applications on Content Server.

What is Beautiful WebForms?¶

Beautiful WebForms is a Module Suite component that enables you to create modern, responsive web forms on OpenText Content Server. It provides a visual form builder, a comprehensive widget library, and powerful integration capabilities that allow you to build sophisticated form-based applications without traditional HTML/JavaScript development.

For a comprehensive overview, see the Beautiful WebForms Views.

Key Components¶

The Beautiful WebForms extension includes:

- **Form Builder** - A web-based IDE with drag-and-drop capabilities for creating form views
- **Widget Library** - A comprehensive set of form widgets and components
- **View Templates** - Pre-configured templates for different form layouts and use cases
- **Custom Logic Execution Hooks (CLEH)** - Server-side scripting hooks for form lifecycle events
- **Smart View Integration** - Embed forms directly in Smart View perspectives
- **Content Script Integration** - Seamless integration with Content Script for server-side logic
- **AI-Based Form Builder** - Experimental feature for creating forms using natural language prompts

Quick Start Guide¶

1. Understanding the Basics¶

Start by reading the Beautiful WebForms Views to understand:

- How Beautiful WebForms work
- The form request lifecycle

- Custom Logic Execution Hooks (CLEH)
- Grid system and responsive layout

2. Creating Form Objects¶

Learn how to create and manage Beautiful WebForms objects:

- Beautiful WebForms Objects - Creating views and understanding view properties

3. Building Forms¶

Learn how to design and build forms using the Form Builder:

- Form Builder - Learn how to use the web-based IDE to design forms
- Smart Editor (WYSIWYG drag-and-drop)
- Source Code Editor (Velocity template editing)
- AI-Based Form Builder (experimental)

4. Working with Widgets¶

Explore the available widgets and their configurations:

- Widgets - Overview of widgets and their usage
- Widgets List - Complete reference of all available widgets
- Widget Behaviours - Server-side behaviors for widgets

5. Advanced Features¶

Explore advanced capabilities and integrations:

- Smart View Integration - Embedding Beautiful WebForms in Smart View perspectives
- SDK - Creating custom widgets and templates
- Language Reference - Content Script language syntax for form logic

Prerequisites¶

Before you begin, make sure you have:

1. Access to an OpenText Content Server instance with Module Suite (>= 3.3) installed and properly configured
2. Familiarity with the basics of creating objects on Content Server
3. Understanding of the following Content Server objects:
4. Form Template
5. View (HTML, WR Power View)
6. Form

Next Steps¶

Once you're familiar with the basics, you can:

- Create your first form using the Form Builder
- Explore the Widget Library to see what's available
- Learn about Custom Logic Execution Hooks for server-side form processing
- Integrate forms with Smart View for modern user interfaces

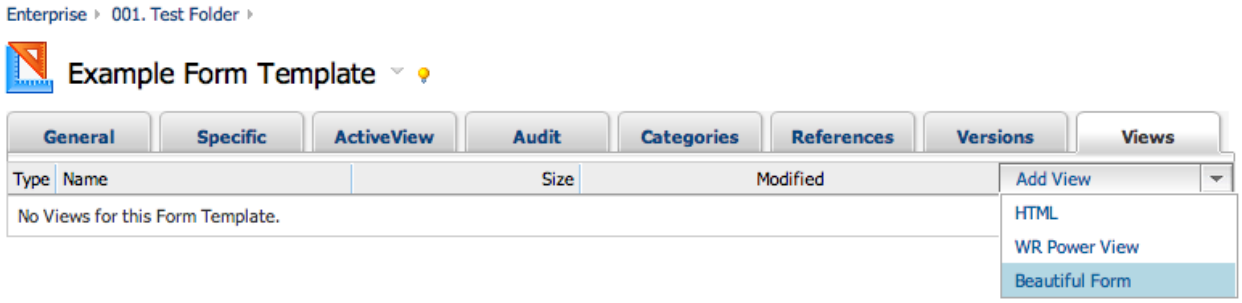
Content Management Object

Beautiful WebForms views are document-class objects on Content Server.

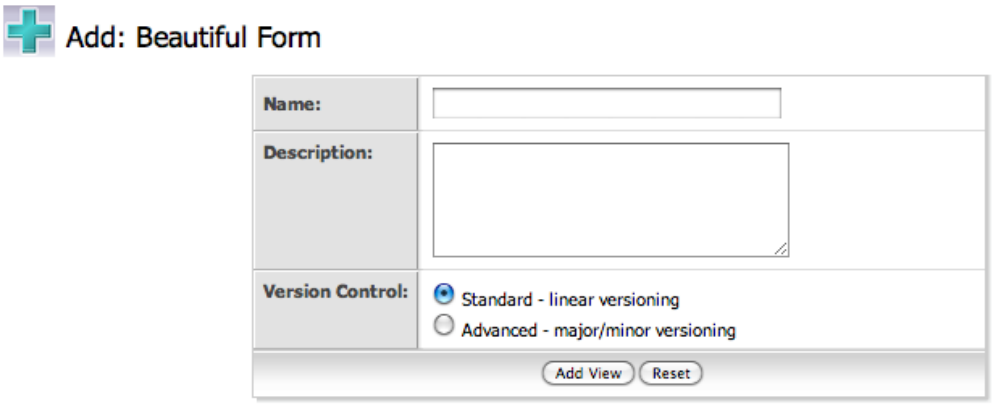
Being standard objects, Beautiful WebForms views comply with Content Server **permissions** model. Upon creation, the object can be edited with the web-based IDE selecting the **'Form Builder'** function in the object function menu.

Creating a Beautiful WebForms View¶

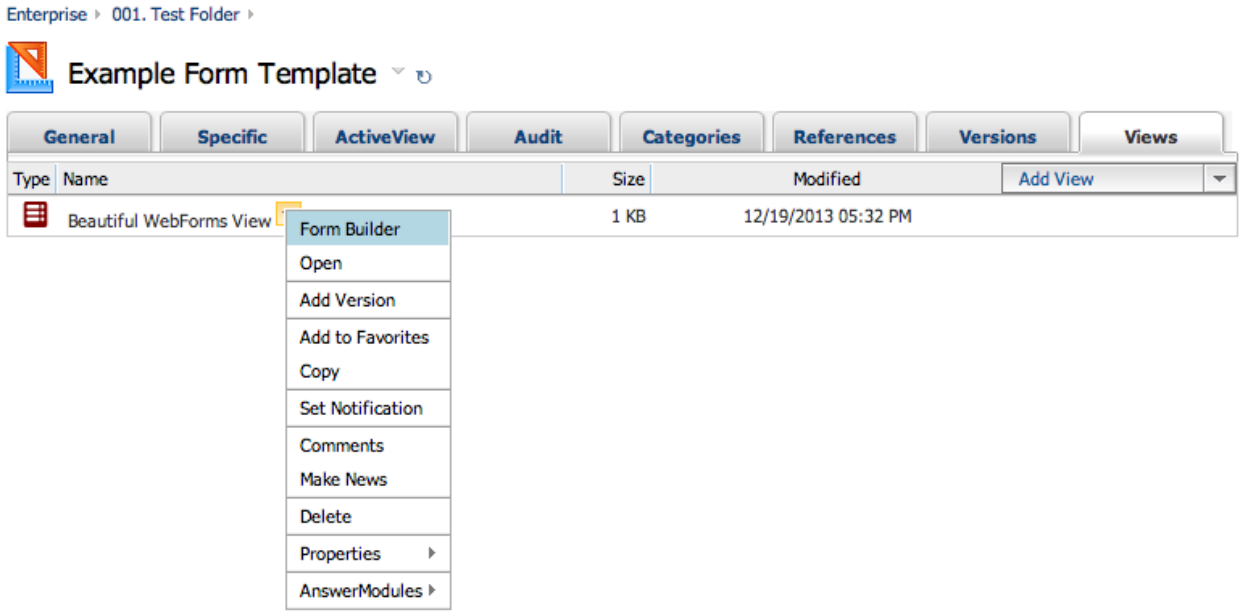
Beautiful WebForms views can be created in the same way as standard html views. In the 'views' tab of the 'form template', an additional 'Beautiful Form' entry will be available in the 'add view' dropdown menu.



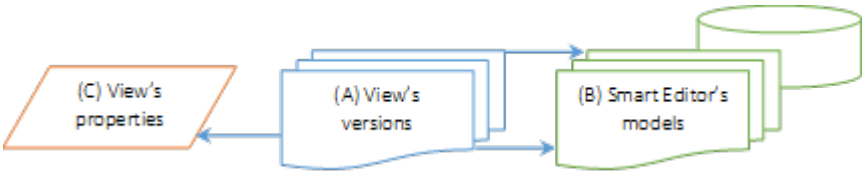
As per standard views, the creation requires a view name be specified. Standard versioning options apply to form views.



Upon creation, the view can be edited with the web-based IDE selecting the **'Form Builder'** option in the object options menu.



Understanding the view object¶



Beautiful WebForms views are much more than simple html-views. They are **active** objects that can be used to create very complex applications. In order to implement all their additional functionalities, Beautiful WebForms views are decorated with a set of information used by the Beautiful WebForms framework for determining how to render, and how to display form's data within them.

In the image above a simplified representation of the information that constitutes a Beautiful WebForms view is highlighted:

- **(A)** View's versions: Beautiful WebForms views are standard FormTemplate's views thus versioned document-class objects. Each version is, in the very end, nothing but a **Velocity** *(http://velocity.apache.org/)* template document (HTML code + template expressions).
- **(B)** For each version created with the FormBuilder's smart-editor the BWF framework archives the smart-editor view's "model" into an internal database table. The smart-editor view's model is constituted by the list of the configurations used for each widget that build the view.
- **(C)** View's properties: Beautiful WebForms views are associated with a set of predefined properties persisted as the object's extended data. These properties are related just to the last view's version.
  The view's predefined properties are:
  1. Form Builder mode used for creating the current view's version (either "source code" or "smart editor")
  2. The list of static "css" view's dependecies dynamically determined on the basis of the widgets used to build the view
  3. The list of static "javascript" view's dependecies dynamically determined on the basis of the widgets used to build the view
  4. The number of view's columns
  5. The identifier of the library of widgets used to build the view
  6. The ID of the view template (if any) associated to the view

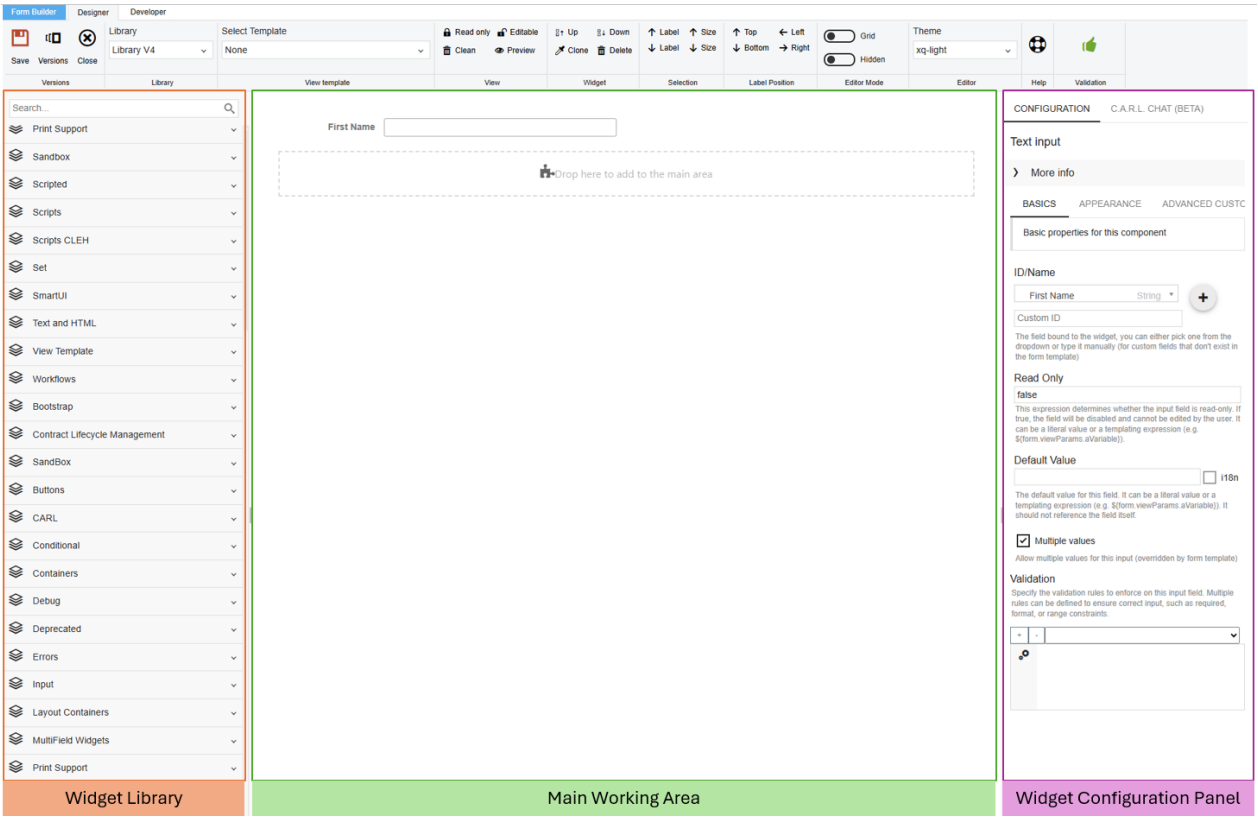
The Form Builder is the privileged IDE for Beautiful WebForms. On the **first load** of an empty view, the Form Builder will initialize it with a default input widget for every field in the form template. The view will then be available for further editing.

Layout¶

The IDE is composed of a set of areas and controls, with different purposes.

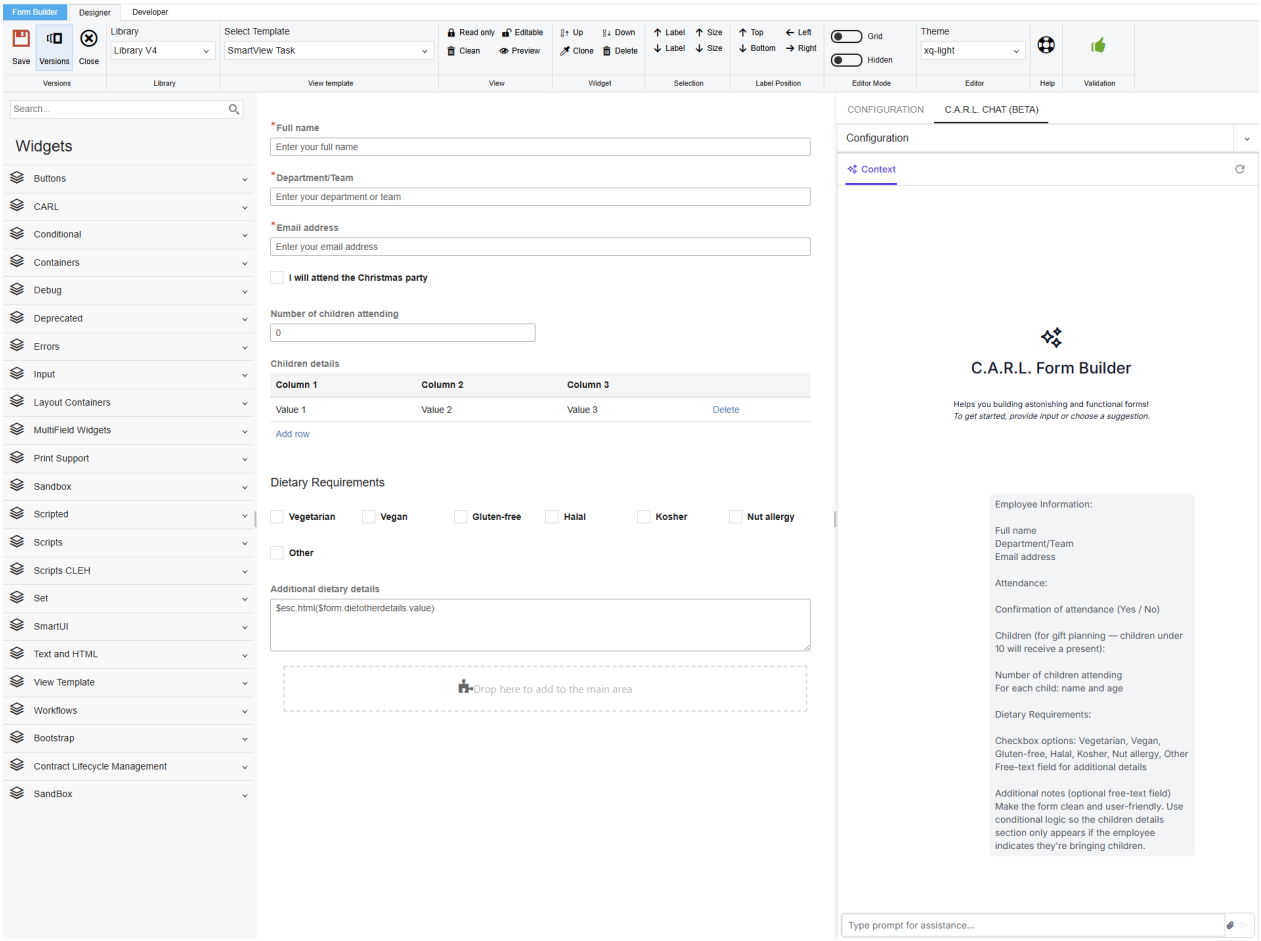
- The **Main Working Area** shows a preview of the current form view, with the available input fields
- The **Widget Library** (on the left) features a set of predefined widgets, which can be easily dragged and dropped in the working area

- The **Widget Configurator panel** (on the right) is linked to the widget currently selected in the main working area



AI-Based Form Builder¶

The AI-Based Form Builder is an experimental feature that enables designers to create forms using natural language prompts. Instead of manually dragging and dropping widgets, you can simply describe the form requirements, and C.A.R.L. (the AI assistant) will automatically generate the form for you.



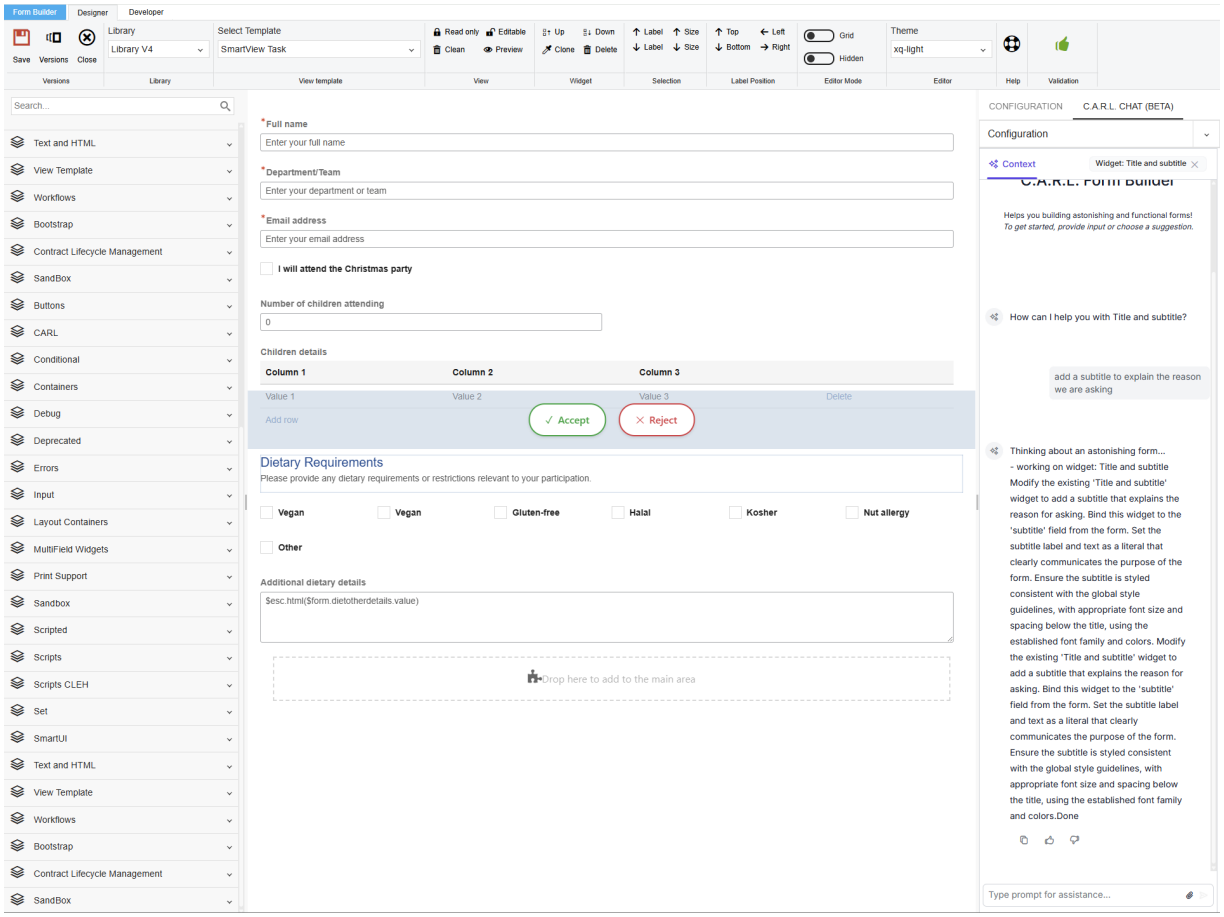
How It Works¶

The AI-Based Form Builder uses an **agentic workflow architecture** that processes your requests through a sophisticated multi-agent system:

1. **Coordinator Agent**: When you submit a request, a coordinator agent first analyzes it and creates an execution plan. This plan determines what needs to be done to satisfy your requirements, including:
2. Identifying which form fields need to be created (if "Allow Creating New Fields" is enabled)
3. Determining which widgets are most appropriate for each field
4. Organizing the form layout and structure
5. **Widget Agents**: Once the plan is determined, specialized widget agents work in parallel to configure each widget according to the coordinator's plan. Each widget agent is dedicated to a specific widget type and handles its configuration independently.

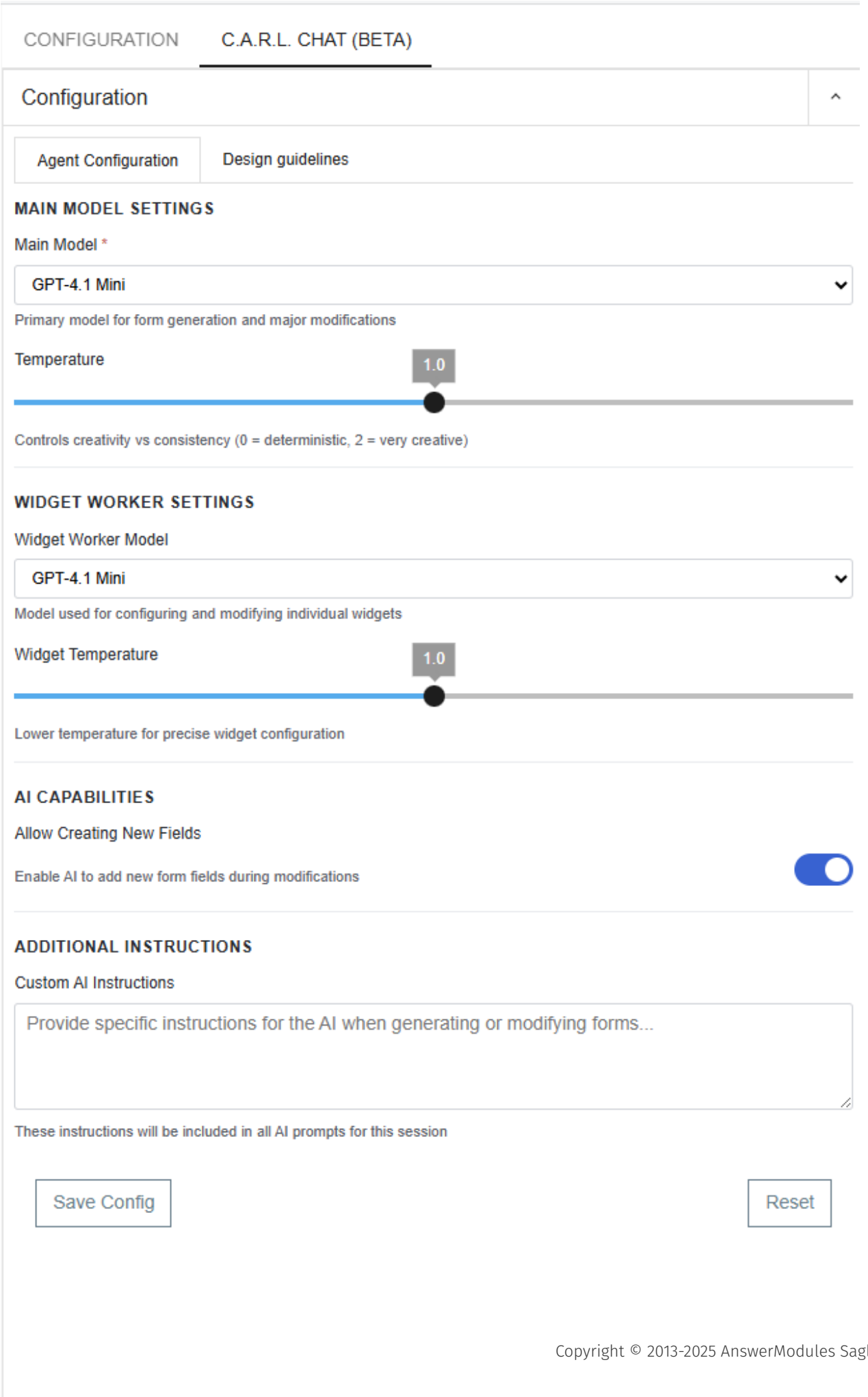
User Confirmation Required

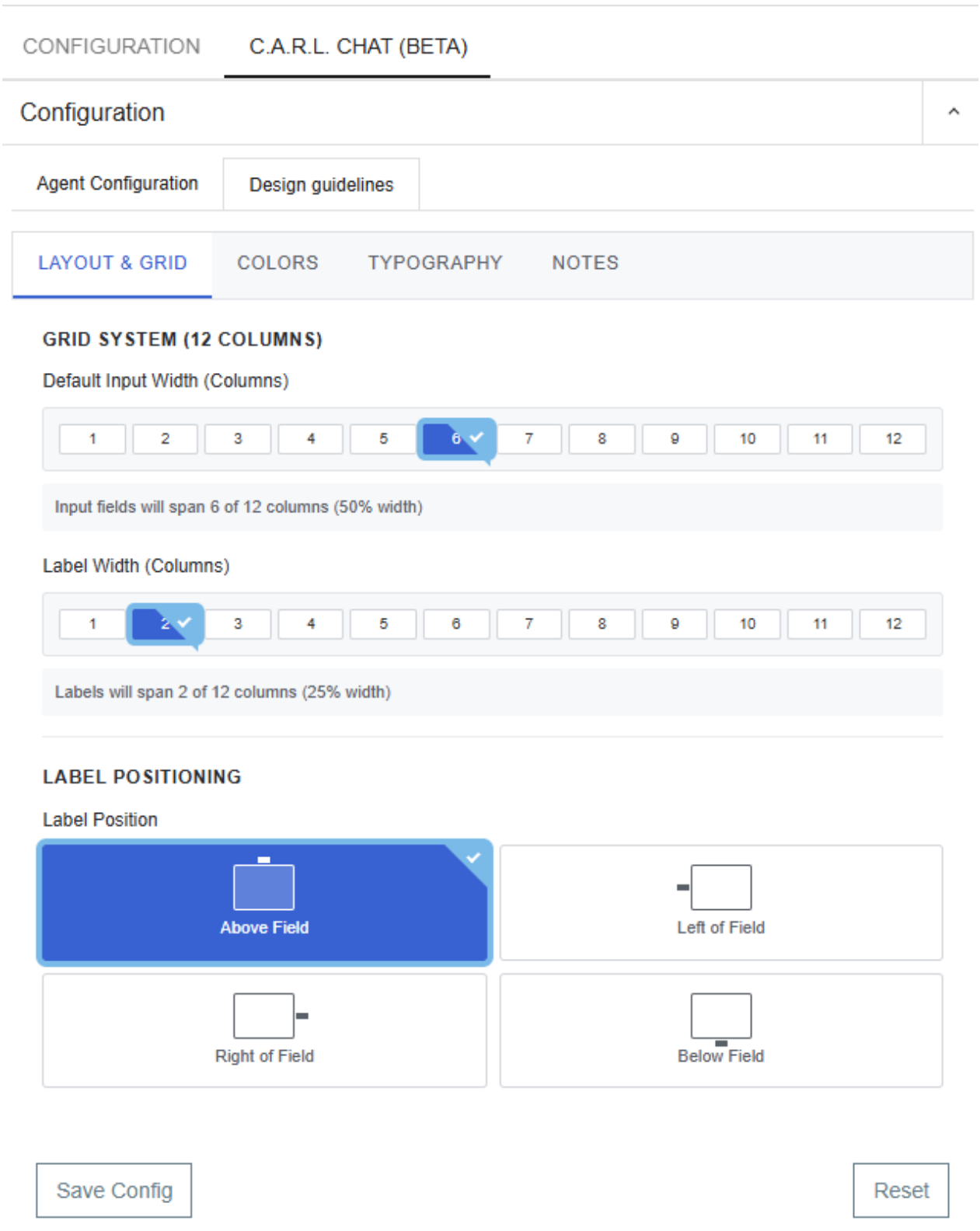
Every change made by the AI requires your explicit confirmation before being applied to the form. This ensures you maintain full control over the form design and can review all AI-generated modifications before they are implemented.



Key Features¶

- **Natural Language Form Creation**: Simply describe what you want in the form, and the AI will create it automatically
- **UI Guidelines Support**: You can provide UI guidelines to guide the AI's design decisions, ensuring the generated forms match your design standards
- **Agent Configuration**: Fine-tune the AI's behavior through a dedicated configuration panel that allows you to adjust various agent parameters





Allow Creating New Fields¶

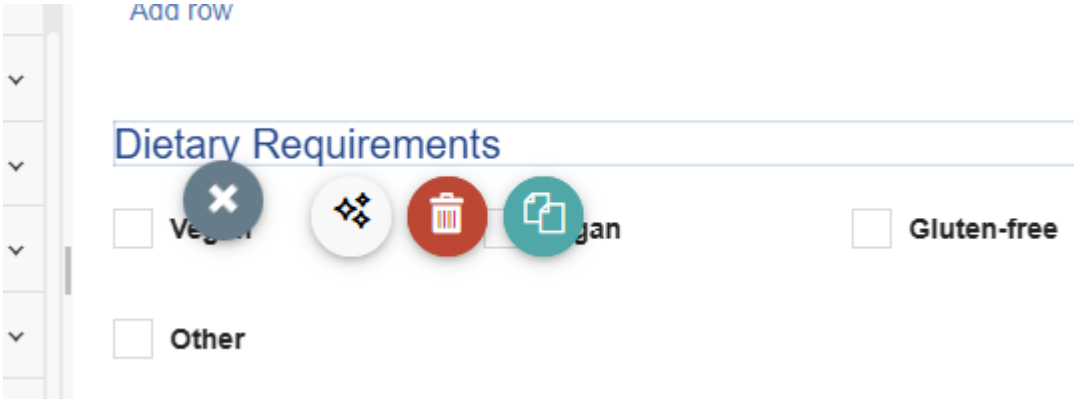
The **"Allow Creating New Fields"** feature controls whether the AI can modify the form template structure:

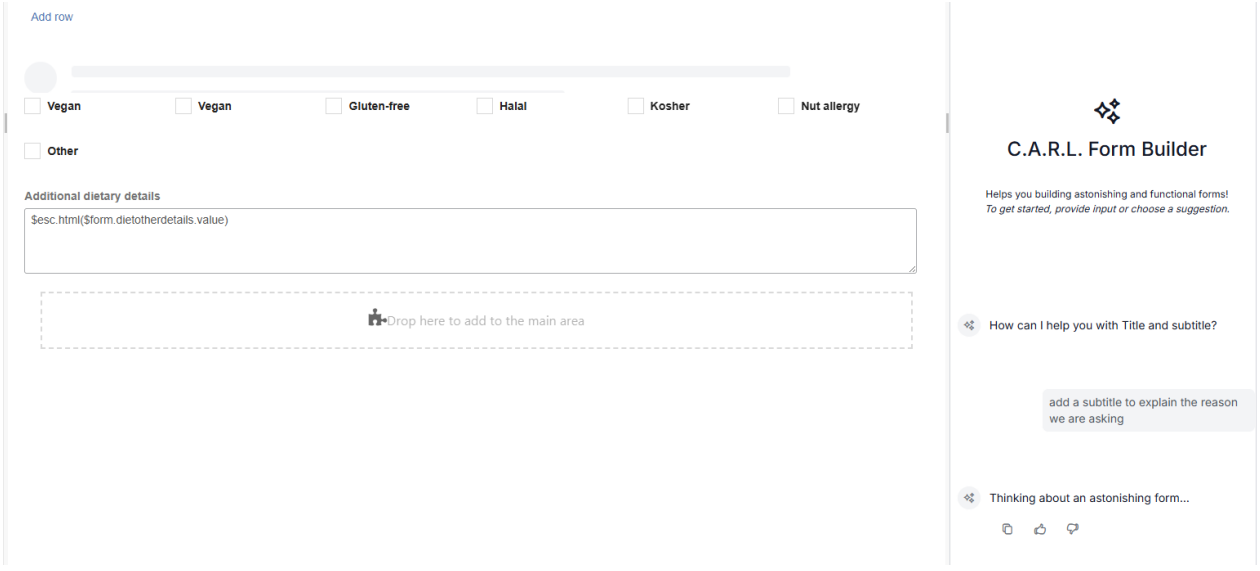
- **Enabled**: The AI can automatically create any fields required for the form based on your description. This gives maximum flexibility and allows the AI to build complete forms from scratch.
- **Disabled**: The AI will only work with existing form template fields and won't modify the template structure. Use this mode when you want to preserve the existing field structure and only modify widget configurations.

Single Widget Configuration¶

In addition to creating entire forms, you can also use the AI feature to configure individual widgets. When you select a specific widget and interact with C.A.R.L., the AI will focus solely on that widget's configuration without modifying the rest of the view. This allows you to:

- Refine individual widget settings using natural language
- Get AI assistance for specific widget properties
- Make targeted improvements without affecting other form elements





Context Support¶

The AI-Based Form Builder supports adding files and images to the context, enabling the AI to:

- Understand visual requirements from reference images
- Process documentation or specifications provided as files
- Generate forms that match design mockups or examples

This context-aware capability allows for more accurate form generation based on visual and textual references.

> **C.A.R.L. Required**
>
> The AI-Based Form Builder is an experimental feature that requires C.A.R.L. integration to be enabled and properly configured on your system. Without C.A.R.L., this feature will not be available.

Developer Guide: Editor Overview¶

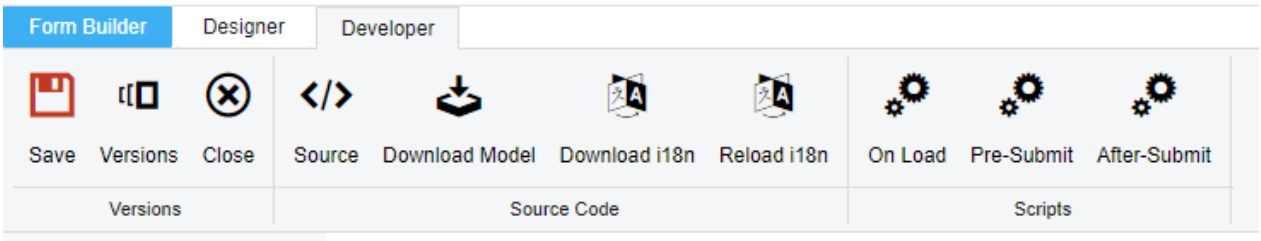
Main Area Functionality¶

The Main Area of our software offers two distinct editing modes:

1. Smart Editor:
    - A WYSIWYG (What You See Is What You Get) drag-and-drop editor.
    - Enables form creation without writing any code.
    - Ideal for quick and intuitive design.
2. Source Code Editor:
    - A text-based editor for modifying Velocity code automatically generated by the Smart Editor.
    - Offers detailed control over the form's code.

By default, the **Smart Editor** is active.

To switch to the **Source Code Editor**, use the *Source* button located in the Developer tab.



Editor Exclusivity¶

- The Smart Editor and Source Code Editor are mutually exclusive; both cannot be active simultaneously.
- Any changes made in the Source Code Editor are **not** preserved if the form is later modified in the Smart Editor.

Check out your notifications

```
The Source Code Editor provides a notification when switching back to the Smart Editor to re
```

Do not use the Source Code editor to modify your view

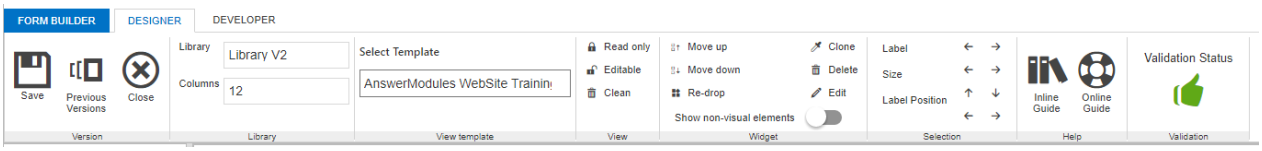
```
We recommend avoiding modifications directly in the Source Code Editor. Instead, consider the

- **Custom HTML Widget**: Allows for specific HTML element customization.
- **Creating a New Widget**: Design your own widget for unique functionality.
- **Modifying an Existing Widget**: Adjust existing widgets to suit your needs.
```

Shortcuts¶

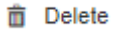
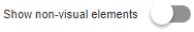
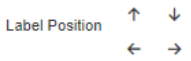
The following keyboard shortcuts are available while using the editor:

Shortcut	Description
Ctrl + S	Save the current view (add a new version)
Ctrl + Canc	Delete the selected widget(s)
Ctrl + B	Clone the selected widget(s)
Shift + Left Ar.	Reduce the label's dimension for the selected widget(s)
Shift + Right Ar.	Increment the label's dimension for the selected widget(s)
Ctrl + Left Ar.	Reduce the dimension for the selected widget(s)
Ctrl + Right Ar.	Increment the dimension for the selected widget(s)
Ctrl + Mouse sel.	Select multiple widgets
Ctrl + Space	In sourcecode editor - show the code autocompletion hints
Ctrl + H	In sourcecode editor - Toggle the online Help window
F11	In widget's configuration panel – Maximize editor (full-screen mode)

Top Bar controls (DESIGNER)¶



Command	**Description**
Versions	
Save	**Save** the view (adds a new version)
Versions	Open the object's **Versions** tab
Close	Close the FormBuilder
Libary	
Library	**Selects** the widgets' library to use for creating the view
Columns	Configures the number of columns in the view layout. In order to take effect, requires to save the view & reload the editor window
View template	
Select Template	The View's template associated with the form can be selected with the dropdown menu, or, as an alternative, selecting a suitable document from Content Server.
View	
Read only	Switch the whole view between Read Only and Editable mode (affects the way input widgets are rendered)
Editable	Switch the whole view between Read Only and Editable mode (affects the way input widgets are rendered)
Clean	Clear the entire working area
Widget	
Move up Move down	Reposition the widget, moving it one step up/down in the form
Re-drop	Pick Up the widget (to drop it elsewhere in view)
Clone	Duplicate the selected widget

Command	**Description**
	Remove the widget from the form
	Open the widget's Configuration Panel
	Toggle the visibility of widgets that are not rendered in the final view (e.g. scripts)
Selection	
	Increase/decrease the size of the widget's label (if available). This option affects the number of columns spanned horizontally by the label.
	Increase/decrease the size of the widget. This option affects the number of columns spanned horizontally by the whole widget (including the label, if present).
	Change of the widget's position. This option affects the number of columns spanned horizontally by the whole widget (including the label, if present).
Help	
	Access the module's online **guide** and the **support portal**
Validation	
	**Red label**: The view failed the validation and most likely will fail to compile
	**Green label:** The view is **well-formed**

Widget Scope

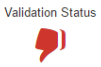
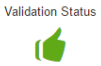
To enable the Widget Scope options in the menu, simply right click on the target widget in the working area.

Columns

When switching the number of columns, save & reload the page editor to force reload of all widgets in the working area

Top Bar controls (DEVELOPER)¶

Command	**Description**
Versions	
Save	**Save** the view (adds a new version)
Versions	Open the object's **Versions** tab
Close	Close the FormBuilder
Source code	
Source Code	**Opens** the view's source code editor
Download Model	Downloads the view's current model to be used for creating a new widget
Download i18n	Downloads the view's localization file
Reload i18n	Reloads all the available localization files
Scripts	
On Load	**Opens** the On-load CLEH Content Script Editor
Pre-Submit	**Opens** the Pre-submit CLEH Content Script Editor
After-Submit	**Opens** the On-submit CLEH Content Script Editor
Validation	
	Red label: The view failed the validation and most likely will fail to compile

Command	**Description**
	
	**Green label:** The view is **well-formed**

Editing source code

View's versions created editing directly the source-code editor can't be further modified with the FormBuilder's smart-editor. If you switch from source-code editor to smart-editor any changes applied modifying the source code will be lost.

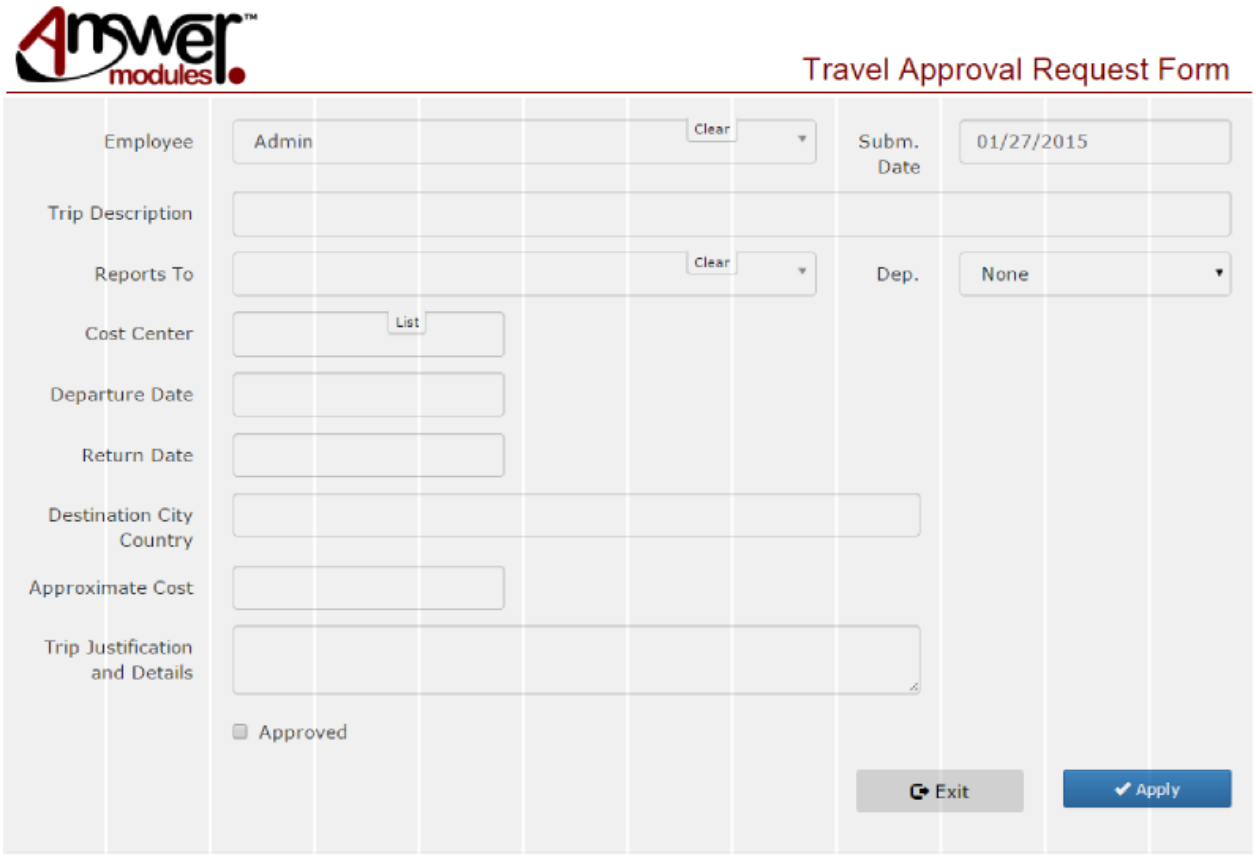
Building views

Understanding the grid system¶

In order to understand some of the features presented in the next sections, it is necessary to introduce the concept of **Grid System**, which has been adopted in the Beautiful WebForms Form Builder and views.

When creating or modifying a Form view, all of the widgets in the view appear neatly aligned to each other. The widgets can be modified in size only in discrete steps: that is, each widget can be assigned a size from a set of predefined options. When the view is presented to the user, the actual size of the widget will be proportional to the selected value.

To understand the logic behind this behaviour, you can imagine the Form fieldset area as if it was divided in a fixed number of columns (12 by default). By forcing each widget to span over a whole number of columns, we keep the overall layout of the form clean and tidy, eliminating the effort that is usually required to fine-tune the alignments and spacings. To better understand this concept, please take a look at the following image.



Additionally, the technology used for the grid layout is **responsive**. The form will automatically adjust to the size of the screen in which it is viewed, degrading gracefully in case of screen of small size.

Understanding the Beautiful WebForms request life-cycle¶

Beautiful WebForms implement a slightly different lifecycle if compared to standard forms, thanks to their custom submission mechanism.

How incoming requests are processed¶

Beautiful WebForms are managed through a dedicated endpoint. Upon submission, the underlying engine performs server side validation. Only after successful validation, the form data is eventually submitted to Content Server.

The Beautiful WebForms life-cycle management of incoming requests can be schematized in the following steps:

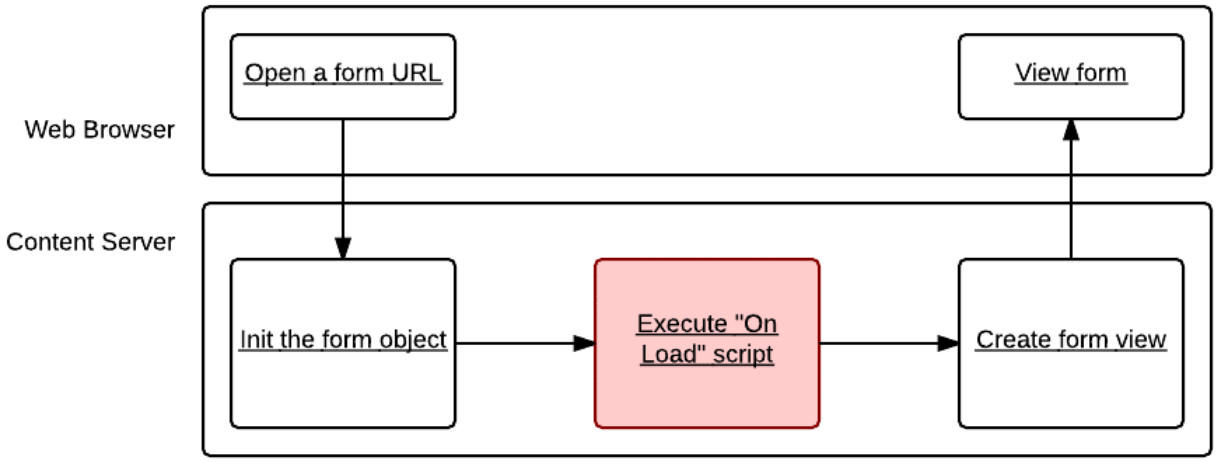
1. Form rendering request: a user requests the form
2. ON LOAD - Custom logic execution hook

3. Form view rendering: the form page is rendered
4. User data input: the user interacts with the form and populates the input fields
5. Form submit action: the user attempts to submit the form data
6. Client side validation: the client side library validates the input fields
7. Actual data submission to Beautiful WebForms endpoint: in case of successful validation, data is submitted to the server
8. Server side validation: the Beautiful WebForms engine performs server side validation on the submitted data
9. PRE SUBMIT - Custom logic execution hook
10. Actual data submission to Content Server: form data is submitted to Content Server
11. POST SUBMIT - Custom logic execution hook
12. A validation error in any of the validation steps would interrupt the flow and return to step 1. Error information would be added to the form view, and used to populate inline error messages.

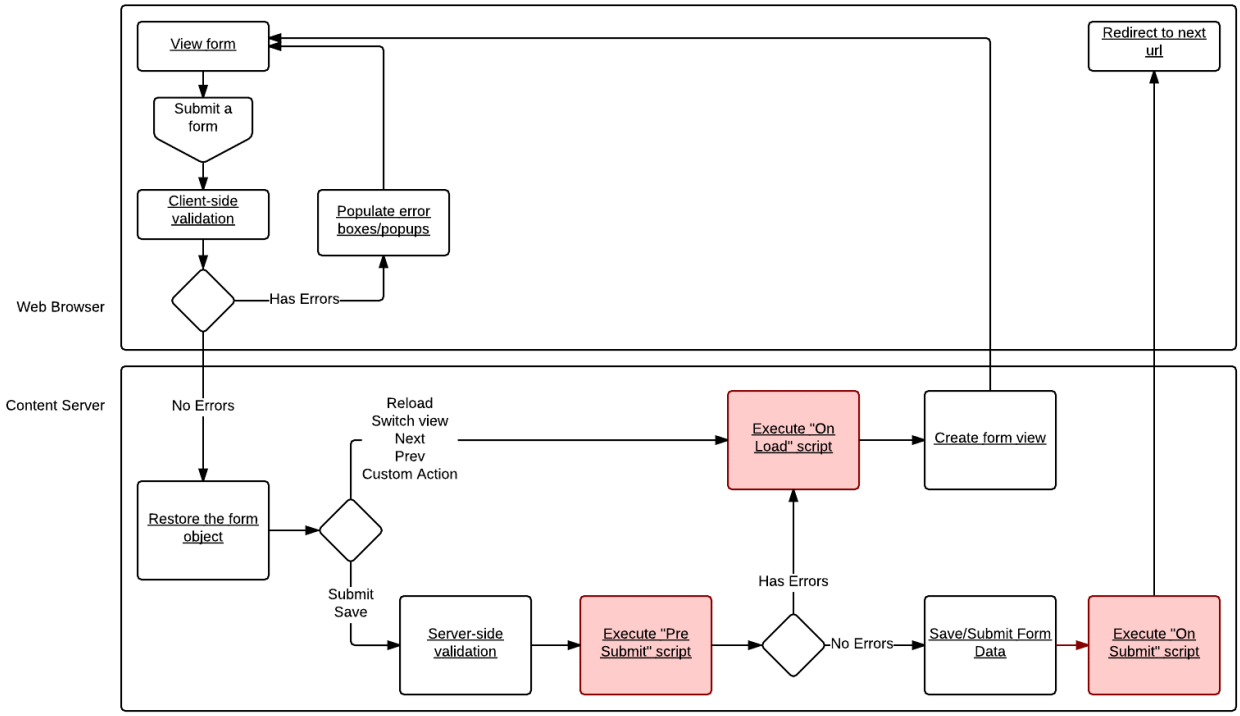
In case of validation errors, the data input by the user is preserved for the following view rendering.

Lifecycle schema¶

The following schema considers a scenario in which a new form is requested by a user:



The following schema is related to a scenario in which the user attempts to submit the form (or otherwise performs an action that triggers a round trip to the server):



Custom Logic Execution Hooks (CLEH)¶

In the two schemas above, there are several highlighted boxes that represent Custom Logic execution hooks. That is, steps in which it is possible to add customized business logic, in the form of Content Script code.

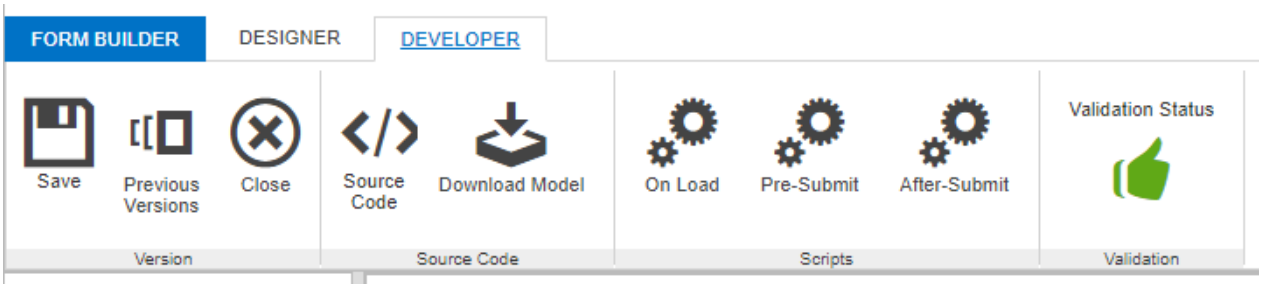
The scripts are:

- **ON LOAD** view Content Script: this is the typical hook for prepopulating the form and manipulating the form view
- **PRE SUBMIT** view Content Script: this is the typical hook for extended validation and actions that must be performed before that the data is actually saved
- **POST SUBMIT** view Content Script: this is the typical hook for post submit actions (user notifications, document manipulation on content server, etc.)

> Starting with version 1.7.0, Beautiful WebForms Views have been transformed in container objects. Content Scripts associated to Beautiful WebForms views are standard Content Script nodes in the view container. The nodes are associated to the lifecycle steps *by name*

Throughout the whole process and in all of these scripts, a form object is available in the execution context. This object allows to fetch and manipulate the form data, as well as programmatically add or remove validation errors.

The Content Script objects associated to each execution hook can be accessed and edited through the **Specific Properties** tab of the Beautiful WebForm view object.



The Content Scripts associated with CLEHs are regular Content Script objects. In the Script Context the Beautiful WebForms framework will inject additional items, such as the **form** object, which represents the form that is currently associated to the view.

The form object grant access to the form fields structure and the current values of each field, the form submitted data, the validation rules associated to the form, and provides utilities to manipulate this information.

E.g.

A commonly used function in the "ON LOAD view script" is

```
form.isFirstLoad()
```

The function allows to define actions which are executed only once per form view (the actions are not repeated in case of submission failure - for example, in case of validation errors). Typically, field prepopulation happens here.

The following sections provide information on common tasks that can be performed on the form programmatically in the various Content Scripts.

Managing form fields values¶

The state of the forms can be programmatically accessed and modified through the Content Script Custom Logic Execution Hooks.

In scripts, form field values can be accessed using the following notation:

```
form.*normalizedname*.value
```

where 'normalizedname' is the name of the field after normalization performed by the Beautiful WebForms framework.

Auto completion

Use the CTRL+Space keyboard shortcut to access autocomplete options on the form object. Options include all the fields in the form.

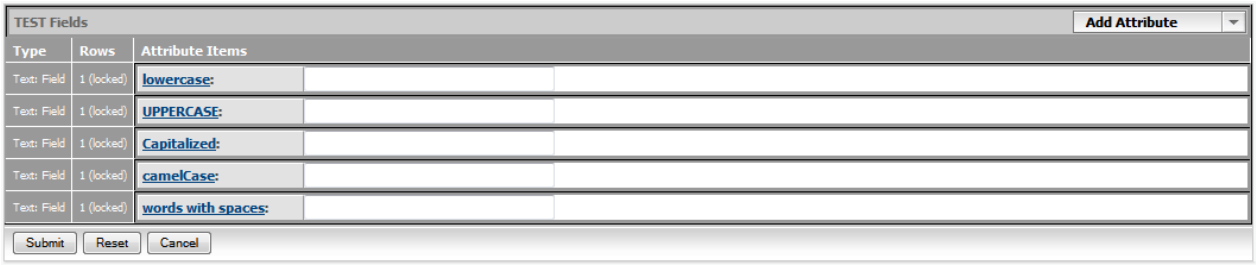
The rules applied when normalizing field names are:

- the only admitted characters are alphanumeric characters and whitespaces (using different characters can lead to unexpected behavior)
- all characters are transformed in lowercase
- all characters immediately after a whitespace are transformed in uppercase

As a rule of thumb, it is advised to adopt a naming convention for field names that would be compatible with SQL table column names.

To better understand the concept, consider the following Form Template, containing a few fields (using different possible naming conventions):

- a field named 'lowercase'
- a field named 'UPPERCASE'
- a field named 'Capitalized'
- a field named 'camelCase'
- a field named 'words with spaces'



The fields can be accessed in a script as follows:

- 'lowercase': form.lowercase.value
- 'UPPERCASE': form.uppercase.value
- 'Capitalized': form.capitalized.value
- 'camelCase': form.camelcase.value
- 'words with spaces': form.wordsWithSpaces.value

```
form.lowercase.value        = "TEST VALUE A" //Form template field name: lowercase
form.uppercase.value        = "TEST VALUE B" //Form template field name: UPPERCASE
form.capitalized.value      = "TEST VALUE C" //Form template field name: Capitalized
form.camelcase.value        = "TEST VALUE D" //Form template field name: camelCase
form.wordsWithSpaces.value  = "TEST VALUE E" ////Form template field name: words with spaces
```

// Initalize form field values: some examples

form.lowercase.value = “TEST VALUE A” // Form template field name: lowercase

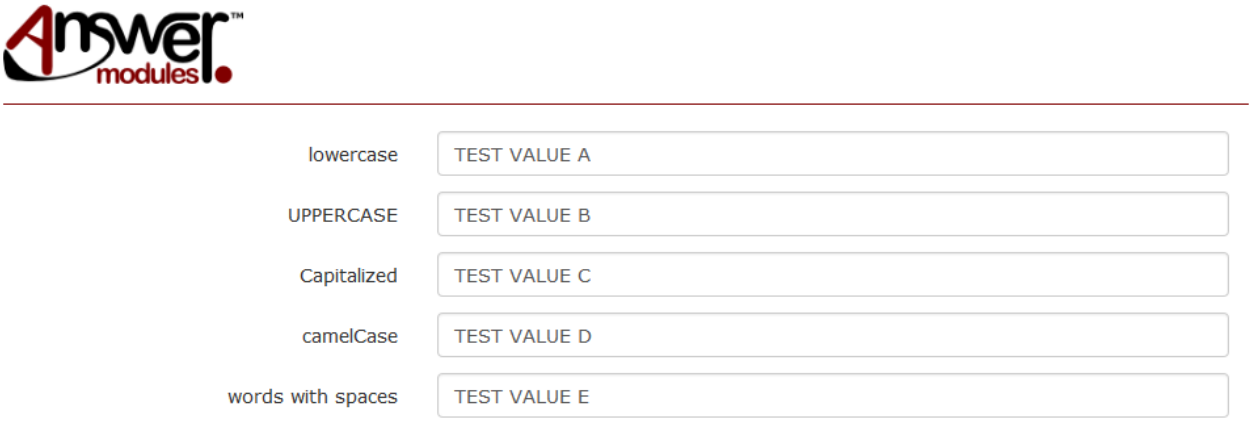
form.uppercase.value = “TEST VALUE B” // Form template field name: UPPERCASE

form.capitalized.value = “TEST VALUE C” // Form template field name: Capitalized

form.camelcase.value = “TEST VALUE D” // Form template field name: camelCase

form.wordsWithSpaces.value = “TEST VALUE E” // Form template field name: words with spaces

The resulting form (after initialization):



Adding and removing values from multivalue fields¶

In case of multi-value fields, it is possible to programmatically add new values (up to the max-values limit)

For each field, multiple values can be accessed directly by index (0-based).

By default, if a field value is accessed without specifying an index, the referenced value is the one with index 0.

form.textvalue.value = "My value" is equivalent to form.textvalue[0].value = "My value"

NOTE: The value at index 0 does not require initialization.

To access values at index > 0:

form.textvalue.addField(1)

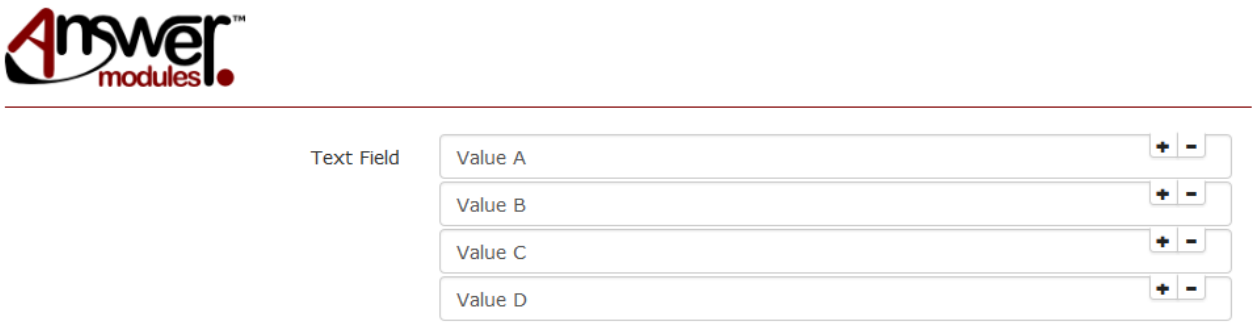
form.textvalue[1].value = "My value"

Example. Field initialization:

```
form.textField.value = "Value A"  // The first field (index:0) is always available. no need to add t
```

```
form.addField("textField", 1) // Additional field's values can be added either through the form obje
form.textField.addField(2)    // or directly on the field

form.textField[1].value = "Value B"
form.textField[2].value = "Value C"

form.textField.addField(3)
form.textField[3].value = "Value D"
```

The resulting form (after initialization):



Form actions¶

An *action* is a piece of server side scripting code that is execute in response of a particular type of request. The action to be performed is identified by the request parameter (`am_Action`) submitted with the form. Another optional parameter (`am_ActionParams`) is sometimes included when specific information is required by the action.

Standard form actions¶

The framework is capable of handling a set of predetermined actions as part of the Beautiful WebForms lifecycle.

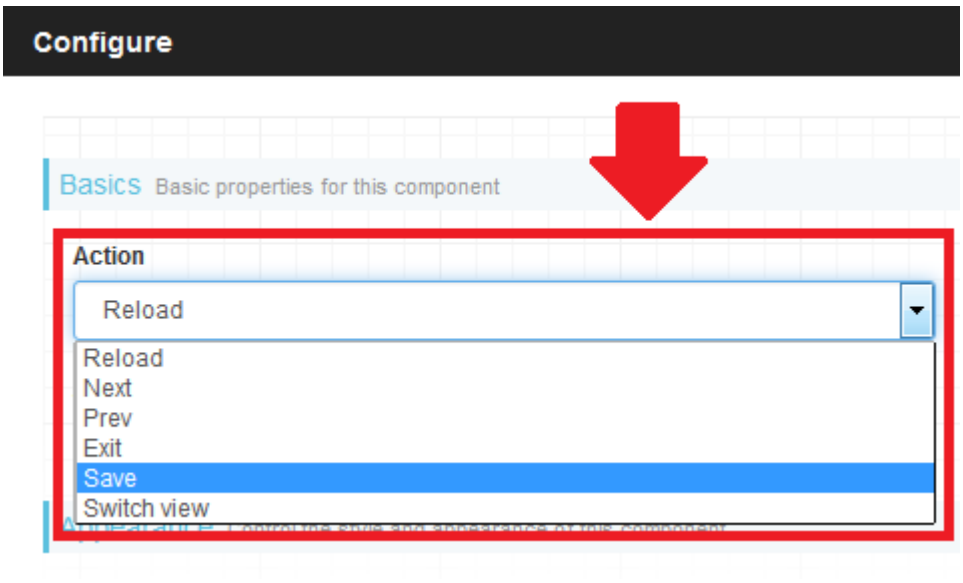
The following are the standard actions managed by the framework:

Action	Description	Action ID (am_Action)	Action parameter (am_ActionParams) usage
Reload	Performs a round trip to the server and re-renders the form view.	`am_reload`	*not required*
Save	Saves the current state of the form, without submitting. *Available in Workflow forms only*	`am_save`	*not required*
Exit		`am_exit`	*not required*

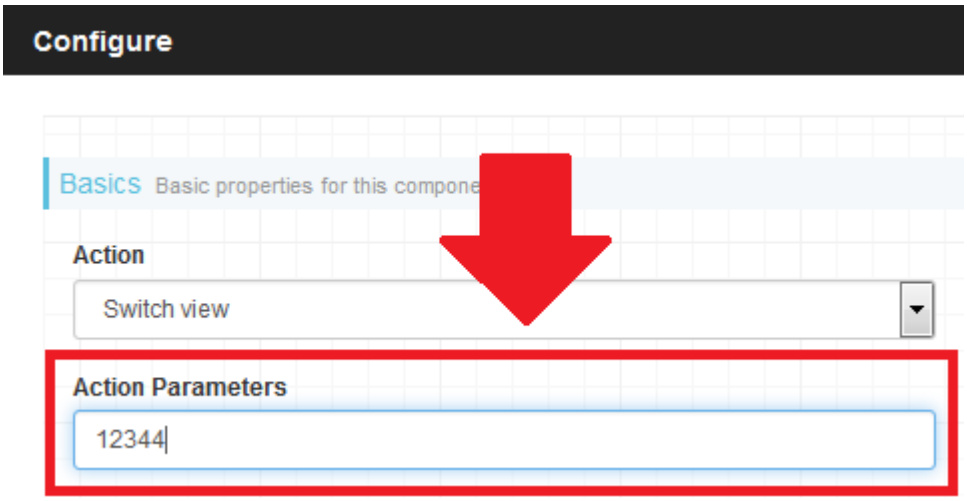
Action	Description	Action ID (am_Action)	Action parameter (am_ActionParams) usage
	Exits without saving modifications to the form data		
Switch View	Switches the view and re-renders the form	`am_switchView`	The ID of the target view
Next	To be used together with "prev" to create a wizard-like experience, enabling the switching forwards through a sequence of different views	`am_wizardNext`	The ID of the next view. Alternatively, the target view can be configured on server side by setting the value of: form.viewParams.am_wizardNextView
Prev	To be used together with "next" to create a wizard-like experience, enabling the switching backwards through a sequence of different views	`am_wizardBack`	The ID of the previous view. If not and a "Next" action was invoked beforehand, the framework will attempt to switch back to that view. Alternatively, the target view can be configured on server side by setting the value of: `form.viewParams.am_wizardPrevView`

Standard form actions can be selected by using the **Standard Action Button** component.

The **Standard Action Button** component can be configured through the configuration panel to select the appropriate action



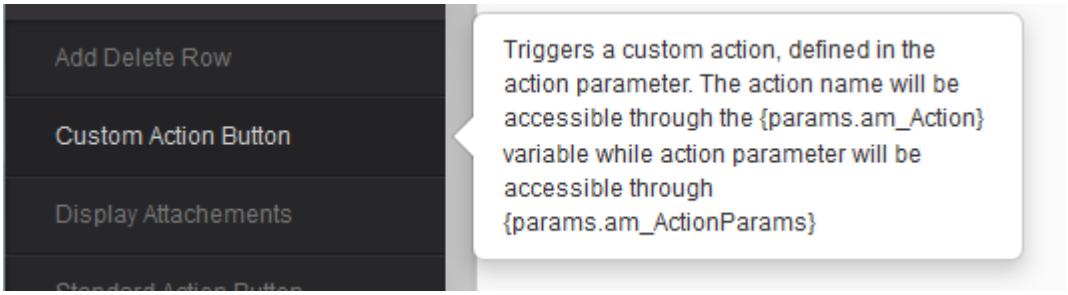
Whenever a parameter is required by the selected action (see above table) the appropriate value can be configured as follows:



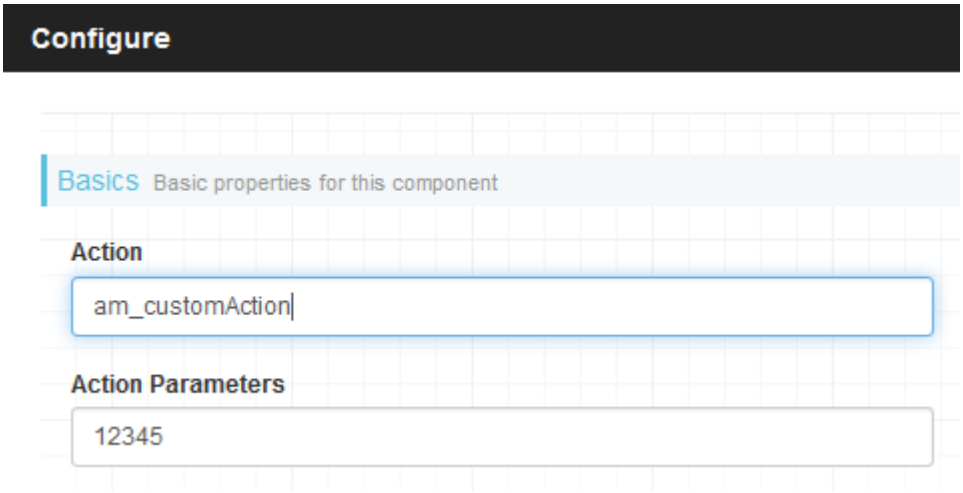
Custom form actions¶

It is also possible to define custom actions when submitting a form. In this case, the custom actions should be handled in the Content Script **Custom Logic Execution Hooks**.

Custom form actions can be selected by using the **Custom Action Button** component.



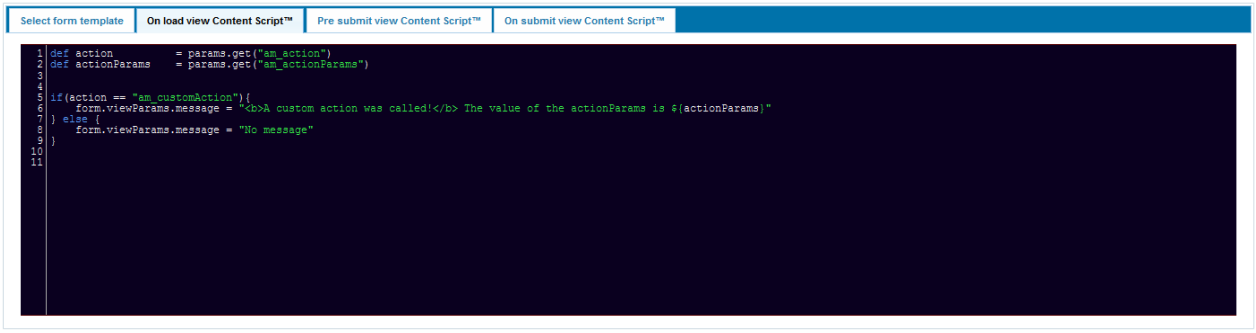
In this case, the configuration panel allows to specify a value for the name of the **action** and the value of the (optional) **actionParams**

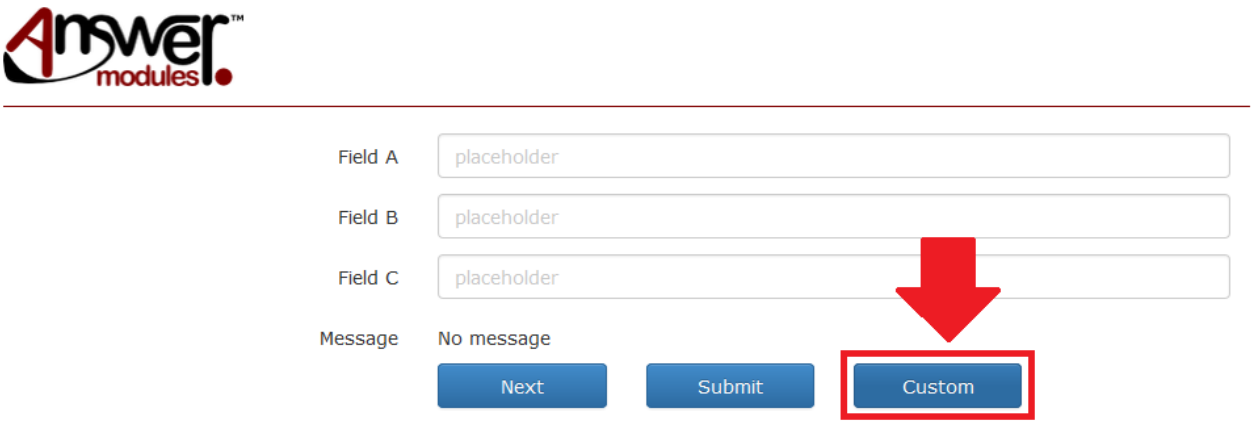


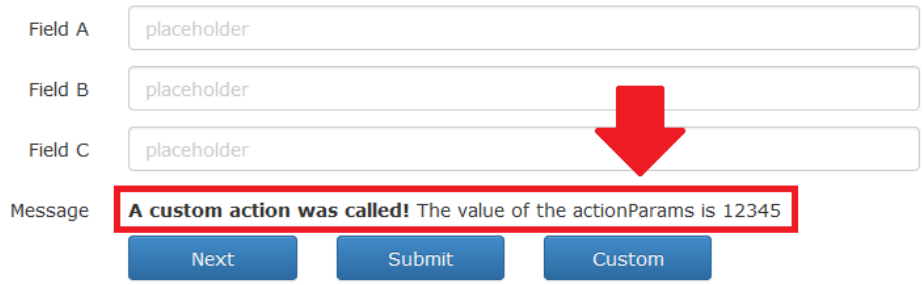
Whenever the button is used, the information related to **action** and **actionParams** will be available in the request params. It can be easily accessed as follows:

```
def action = params.get("am_action")
def actionParams = params.get("am_actionParams")
```

Below is a simple example showing how to use and manage a **Custom Action**:







Invoking an action

It is possible to manually trigger the execution of Actions in cases where the provided Form Components are not sufficient to meet specific needs.

In such cases, the `am_setAction(form, action, actionParams)` javascript function can be used, where:

- **form** is the id of the html form (eg. `form_258191`)
- **action** is the action id (eg. `am_customAction`)
- **actionParams** is the optional value of additional parameters required by the action (eg. '12345')

The following is an example using an HTML button:

```
<button
onclick="am_setAction('form_258191','am_customAction','12345')"
type="submit"> Custom Action Button </button>
```

Attaching Custom information and data to a Beautiful WebForms view¶

ViewParams¶

It is sometimes necessary to bind to the form object additional parameters and values that are not supposed to be stored in form fields. It is the case for parameters that are only needed to control the form page layout: an example is when the HTML template containing the form can be dynamically configured in some of its parts (for example, a title or logo).

To address this need, the **'form'** object is bound to a data map (named **'viewParams'**) which is meant to contain additional parameters that are not supposed to be persisted with the form data.

Entries in the **'viewParams'** map can be set and accessed programmatically as in the following examples.

Example 1. Within a Content Script, set the value of the parameter 'title':

```
form.viewParams.title = "My Form"
```

Example 2. Within a Content Script, read the value of the parameter 'title' and store the value in a variable 'myVar':

```
def myVar = form.viewParams.title
```

Example 3. When accessing the 'viewParams' in an HTML Form Template, the syntax is slightly different, as the templating engine syntax must be used. For example:

```
<h1>$form.viewParams.title</h1>
```

You can include a '!' in your expression in order to avoid printing the output in the rendered HTML in case the value of the variable is not set:

```
<h1>$!form.viewParams.title</h1>
```

Serializable

any object programmatically added to the 'viewParams' map MUST be a serializable object.

ViewParams variables¶

Prior of each view rendering, the Beautiful Form Frameworks injects in the **viewParams** field of the Form object a set of variables. The number and type of these variables depend on the current execution scope. All the variables at the moment of the injection are serialized as String. The table here below summarizes all the possible variables that can be found in the **viewParams** field, indicating for each of them, the original type and name.

Warning

the actual case of the variable names could depend on the underlying database.

List of the variable automatically injected into the ViewParams map

Variable Name	Scope	Original Type
LL_CgiPath	Form, Workflow	String
LL_NextURL	Form, Workflow	String
LL_SupportPath	Form, Workflow	String
LL_UserContact	Form, Workflow	String
LL_UserFirstName	Form, Workflow	String
LL_UserFullName	Form, Workflow	String
LL_UserGroupName	Form, Workflow	String
LL_UserID	Form, Workflow	Integer

LL_UserLastName	Form, Workflow	String
LL_UserLogin	Form, Workflow	String
LL_UserMailAddress	Form, Workflow	String
LL_UserMiddleName	Form, Workflow	String
LL_UserTitle	Form, Workflow	String
MapTask_CustomData	Workflow	Assoc
MapTask_Description	Workflow	String
MapTask_Form	Workflow	Assoc
MapTask_Instructions	Workflow	String
MapTask_Priority	Workflow	Integer
MapTask_StartDate	Workflow	Date
MapTask_SubMapID	Workflow	Integer
MapTask_SubType	Workflow	Integer
MapTask_Type	Workflow	Integer
Map_Description	Workflow	String
Map_Instructions	Workflow	String
Map_SubType	Workflow	Integer
Map_Type	Workflow	Integer
SubWorkTask_DateDone	Workflow	Date
SubWorkTask_DateDue_Max	Workflow	Date
SubWorkTask_DateDue_Min	Workflow	Date
SubWorkTask_DateMilestone	Workflow	Date
SubWorkTask_DateReady	Workflow	Date
SubWorkTask_Flags	Workflow	Integer
SubWorkTask_IterNum	Workflow	Integer
SubWorkTask_PerformerID	Workflow	Integer
SubWorkTask_Status	Workflow	Integer
SubWorkTask_SubWorkID	Workflow	Integer
SubWorkTask_TaskID	Workflow	Integer
SubWorkTask_Title	Workflow	String
SubWorkTask_Type	Workflow	Integer
SubWorkTask_WaitCount	Workflow	Integer
SubWorkTask_WorkID	Workflow	Integer
SubWork_DateCompleted	Workflow	Date
SubWork_DateDue_Max	Workflow	Date
SubWork_DateDue_Min	Workflow	Date
SubWork_DateInitiated	Workflow	Date
SubWork_Flags	Workflow	Integer
SubWork_MapID	Workflow	Integer
SubWork_Project	Workflow	Dynamic
SubWork_ReturnSubWorkID	Workflow	Integer
SubWork_ReturnTaskID	Workflow	Integer
SubWork_Status	Workflow	Integer
SubWork_SubWorkID	Workflow	Integer
SubWork_Title	Workflow	String
SubWork_WorkID	Workflow	Integer
Work_DateCompleted	Workflow	Date
Work_DateDue_Max	Workflow	Date
Work_DateDue_Min	Workflow	Date
Work_DateInitiated	Workflow	Date
Work_Flags	Workflow	Integer
Work_ManagerID	Workflow	Integer
Work_OwnerID	Workflow	Integer
Work_Status	Workflow	Integer
Work_WorkID	Workflow	Integer

Form Components that make use of 'viewParams' values.¶

Various components available in the Form Builder are configurable and require one or more parameters to be programmatically set: these parameters can be made available to the component as values in the **'viewParams'** container variable.

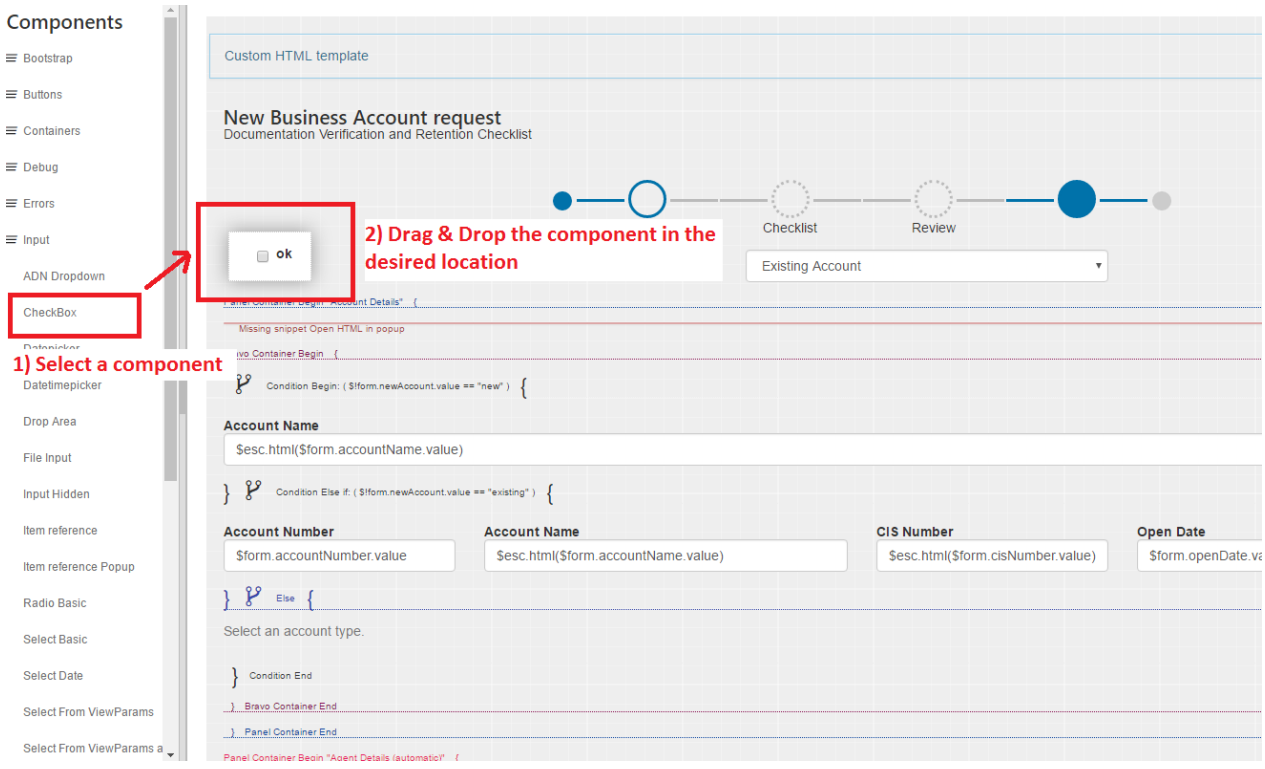
The widgets library¶

The **Widgets library** is an extensible set of form widgets that can be used through the drag & drop visual editor. To simplify the navigation, the widgets are arranged in families of objects with similar functionalities.

> The mapping between form template fields and their default input widget used to initialize Beautiful WebForms Views can be customized by configuring the desired CSFormSnippet in the Content Script Volume.

To add a new widget:

1. Open the widget library group that contains the widget
2. Click on the widget, holding the mouse button down
3. Drag the widget to the desired position in the working area (a highlighted box will appear)
4. Drop the widget in the working area

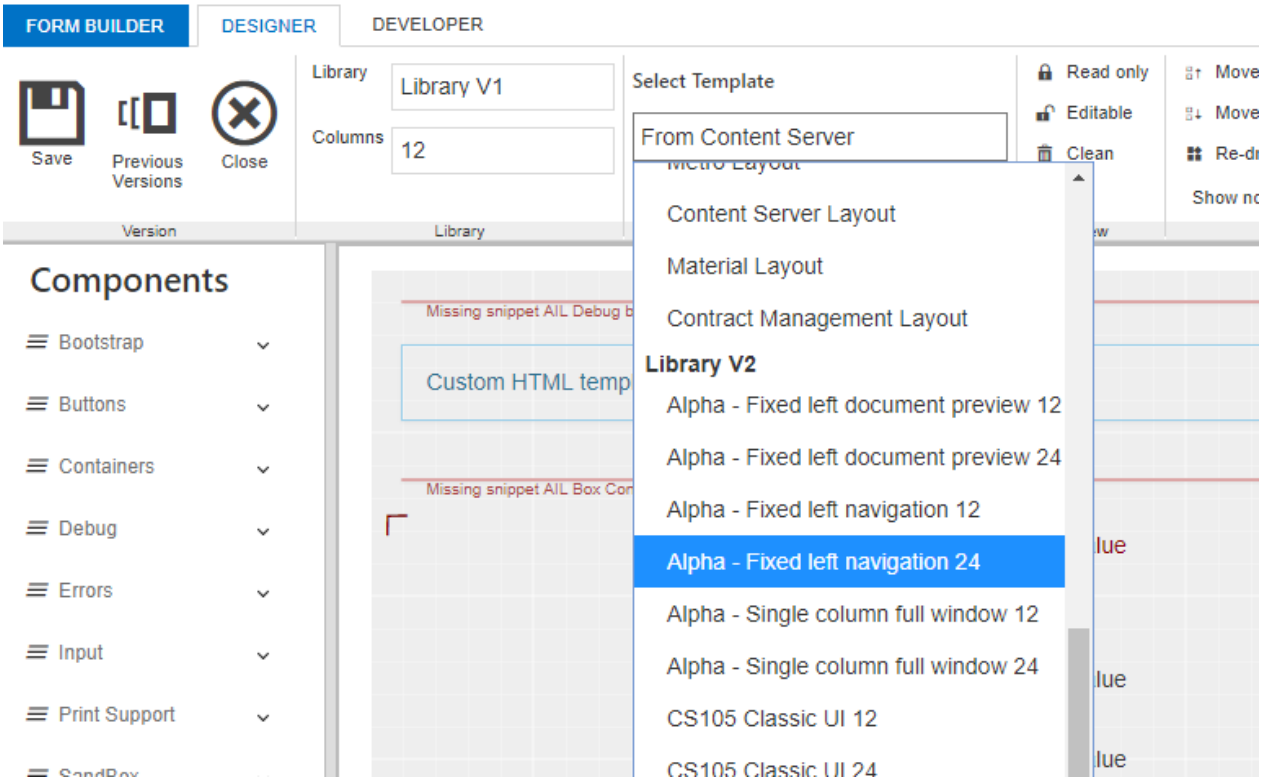


The widget configuration panel¶

When a widget in the Main Working Area is selected, the **Configuration Panel** can be activated through the dedicated menu option or by right-clicking the widget. The content of the panel is specific to the type of widget, and allows to define the widget binding to underlying form fields

(in case of input widgets), as well as how the widget will be rendered, what validation rules will be applied to it, and any other setting that could be necessary for the specific widget.

Beautiful WebForms View Templates¶



The BWF Framework enforces the Model View Controller paradigm, in fact Beautiful WebForms Views (and Templates) are always processed, before being rendered, from the module's internal Templating engine. At rendering time the BWF framework creates (as Model) for the Form View an Execution Context very similar to the one used by the Content Script Engine. The main difference between the two contexts is the presence of the "form" variable that refers to a server side representation of the Form object to which the Form View has been associated. As discussed each BWF View can be associated to a Form Template. At rendering time the framework executes the following operations:

- Substitutes in the Form Template any occurrences of the tag **<am:form />** with the content of the Form View as defined, for example, using the Form Builder
- Evaluates the result of the previous operation with the internal Templating Engine

The most important consequence of the aforementioned rendering procedure is that **any** valid Templating expression present both in the View and in the Template will be evaluated and eventually substituted by the Templating engine. This feature is widely used by default Form Templates and default Form Snippets.

Default Form Templates make use of these characteristics of the framework to slightly change their aspect, resulting behaviors, or more simply to load the most appropriate static resources (i.e. javascript libraries and CSS stylesheets).

For developers convenience the BWF frameworks defines also a set of macro that simplify the creation of new templates or the management of existing one. In the following section the source code of these macro is listed.

Customize the way validation error messages are rendered¶

In order to customize the way validation error messages related to form's fields are displayed you can leverage the Errors *(/working/bwebforms/widgets/#errors_1)* widget in order to override both the javascript (used to render errors on client side) and Velocity (used to render errors on server side) functions in your view.

```
(function(root, factory){
if (typeof csui !== 'undefined' && typeof csui.require === 'function') {
    csui.require(['jquery','underscore','v3/js/am/am_init','v3/js/am/am_ajaxvalidation'], function($
        factory($, _, amui, amform);
    });
}else if ( typeof require === 'function'){
    require(['jquery','underscore','v3/js/am/am_init','v3/js/am/am_ajaxvalidation'], function($,_, a
        factory($, _, amui, amform);
    });
}  else {
    factory(root.jQuery, root.amui);
}
}(this, function($, _, amui, amform) {
```

```
    amform.zcleanFieldValidationError = function (comp){
        var wrapper =comp.closest('.am-form-input-wrap')
        wrapper.removeClass('am-has-error-tooltip')
        wrapper.removeClass('has-error')
        wrapper.data('title', '').attr('title', '');
        try {
            wrapper.tooltip('destroy')
        } catch (e) {
        }
    }

    amform.zcleanFormValidationError = function (form){
        form.find('.help-block.has-error').remove();
        form.find('.am-form-input-wrap').removeClass('has-error');
        form.find('.am-has-error-tooltip').each(
        function() {
            $(this).removeClass('am-has-error-tooltip').data('title', '')
                    .attr('title', '')
            try {
                $(this).tooltip('destroy')
            } catch (e) {
            }
        });
    }

    amform.zdisplayValidationError= function (message, failingElements){
        $(failingElements).each(
                function() {
                    var wrapper = $(this).closest('.am-form-input-wrap')
                    try {
                        wrapper.addClass('am-has-error-tooltip').addClass(
                                'has-error').attr(
                                'title',
                                ((wrapper.data('title') != undefined) ? wrapper
                                        .data('title') : '')
                                        + ' ' + message);
                        wrapper.tooltip('destroy')
                        wrapper.tooltip()
                    } catch (e) {
                    }
                });
    }
}));
```

```
#macro( showErrors $field )
<script>
(function(root, factory) {
    if (typeof csui !== 'undefined' && typeof csui.require === 'function') {
        csui.require(['jquery','v3/js/am/am_init','underscore','regula'], function($,amui,underscore
            factory($, amui, _, regula);
        });
    }else if (typeof require === 'function') {
        require(['jquery', 'v3/js/am/am_init', 'underscore', 'regula'], function ($, amui, _, regula
             return factory($, amui, _ ,regula);
        });
    } else {
        factory(root.jQuery, root.amui, root._, regula);
    }
}(this, function($, amui, _, regula) {
       #if($field.getValidationStatus().size() gt 0)
       amui.registerInitWidgetCallback(function(){
            $('#$field.id').data('title','');
            #foreach ($error in $field.getValidationStatus() )
                    $('#$field.id').data('title', $('#$field.id').data('title')+' $error.validationE
            #end
            var wrapper = $('#$field.id').closest('.am-form-input-wrap');
            try{
```

```
                wrapper.tooltip('destroy')
            }catch(e){
            }
                        wrapper.addClass('am-has-error-tooltip')
                        .data('title', $('#$field.id').data('title'))
                        .attr('title', $('#$field.id').data('title'))
                        .tooltip()
                        .addClass('has-error');
        });
        #end
}));
</script>
#end
```

Display errors in Smart View¶

In order to be compliant with the way SmartView displays error messages the following overrides can be utilized

```
(function(root, factory){
if (typeof csui !== 'undefined' && typeof csui.require === 'function') {
    csui.require(['jquery','underscore','v3/js/am/am_init','v3/js/am/am_ajaxvalidation'], function($
        factory($, _, amui, amform);
    });
}else if ( typeof require === 'function'){
    require(['jquery','underscore','v3/js/am/am_init','v3/js/am/am_ajaxvalidation'], function($,_, a
        factory($, _, amui, amform);
    });
}  else {
    factory(root.jQuery, root.amui);
}
}(this, function($, _, amui, amform) {

    amform.zdisplayValidationError= function(message, failingElements){
        $(failingElements).each(
            function() {
                var wrapper = $(this);
                try {
                    wrapper.addClass("am-smartui-error");
                    wrapper.closest('.am-form-input-wrap').append("<div class='amsmartui-help-block

                } catch (e) {
                    //jquery compatibility
                }
            });
    }

    amform.zcleanFieldValidationError=function(comp){
        var wrapper =comp
        wrapper.removeClass('am-smartui-error')
        wrapper.closest('.am-form-input-wrap').find(".amsmartui-help-block").remove();

    }

    amform.zcleanFormValidationError = function(form){
        form.find('.help-block.has-error').remove();
        form.find('.am-form-input-wrap').removeClass('has-error');
        form.find('.am-smartui-error').each(
            function() {
                $(this).removeClass('am-smartui-error').closest('.am-form-input-wrap').find(".amsmar
            });
    }
}));
```

```
#macro( showErrors $field )
<script>
(function(root, factory) {
    if (typeof csui !== 'undefined' && typeof csui.require === 'function') {
        csui.require(['jquery','v3/js/am/am_init','underscore','regula'], function($,amui,underscore
            factory($, amui, _, regula);
        });
    }else if (typeof require === 'function') {
        require(['jquery', 'v3/js/am/am_init', 'underscore', 'regula'], function ($, amui, _, regula
             return factory($, amui, _ ,regula);
        });
    } else {
        factory(root.jQuery, root.amui, root._, regula);
    }
}(this, function($, amui, _, regula) {
       #if($field.getValidationStatus().size() gt 0)
       amui.registerInitWidgetCallback(function(){
           var wrapper = $('#$field.id');
           wrapper.addClass("am-smartui-error");
           #foreach ($error in $field.getValidationStatus() )
           wrapper.closest('.am-form-input-wrap').append("<div class='amsmartui-help-block form-cont
           #end

       });
       #end
}));
</script>
#end
```

performances-tips

Widgets

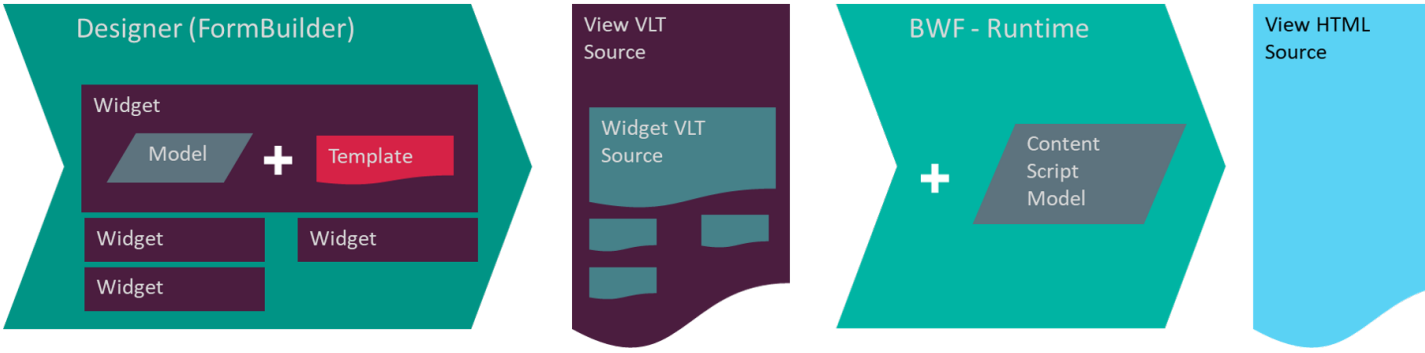
Beautiful WebForms Widgets¶

Beautiful WebForms Widgets are the base units a View is composed of (a View is in fact nothing but a collection of Widgets). Beautiful WebForms Widgets are implemented by Module Suite Template objects of type *Beautiful WebForm Snippet* stored under the **CSFormSnippets** folder in the Content Script Volume *(/administration/csvolume/)*.

Widgets are defined by a Model and a Template.

View's Widgets templates and their models are evaluated by the Form Builder[1] to produce the intermediate View Velocity Template Document (VVTD).

At runtime (when a WebForm is rendered) the Beautiful WebForm MVC framework evaluates the VTD against a Content Script Model to produce the final WebForm HTML page.



Model and Template¶

The Widget model is implemented in the form of a Javascript object while the template is implemented in the form of an Handlebars *(https://handlebarsjs.com/)* template. The template might contain a set of partials *(https://handlebarsjs.com/guide/partials.html#partials)* defined by Module Suite Template objects of type *Content Script Snippet* stored under the **CSSystem** folder in the Content Script Volume *(/administration/csvolume/)*, partials can be identified because their name is prefixed by the ***Partial*** keyword.

Below an example of a Widget Model and template:

◉○
ModelTemplate

```
{
  "fields":{
    ...
    "h_base"  :{"title":"Basics","type":"_help","help":"oh_baseProperties"},
    "fieldA":{"label":"A Field Label","type":"input","value":"","help":"Field's help message", "i18n
    ...
  }
  ,"title":"My Widget"
  ,"help":{"value":"oh_textInput"}
  ,"order":["fieldA", "fieldB"]
  ,"jsdependencies":[]
  ,"cssdependencies":[]
  ,"nonRendableWidgets":false
  ,"columns":true
  ,"binding":true
  ,"style":true
  ,"validation":true
  ,"readonly":true
  ,"container":false,
  ,"rendered": true
}
```

```
69
70  {{> _renderedOpen}} {{!-- Manages the "Show-if" configuration option (creates the VTL expressio
71  {{#*inline "_componentClass"}}am-form-text-input{{/inline}}

    {{#if label}}^
        {{> _labelLeft}}
        <div class="{{> _colSize}} {{>_componentClass}} {{add_class}}" {{>_amWID}}>  {{!-- {{> _col
        {{> _labelTop}}

    {{else}}
        <div class="{{> _colSize}} {{>_componentClass}} {{#unless label}}{{#if required}}am-form-re
    {{/if}}

            {{#if render}}
            #foreach( $rowField in {{id}} )
            {{> _defaultValue }}
            {{/if}}

            <div class="am-form-input-wrap"  {{> _popover }} >

                {{#if render}}
                #if({{{readonly}}})

                <p class="{{#if bold_body}}am-form-bold{{/if}}" >
                    <span class="form-control-static"> $esc.html({{id}}.value)</span>
                </p>

                #else

                {{/if}}
                <input id="{{id}}.id"
                      name="{{id}}.id"
                      value="{{#if render}}$esc.html({{id}}.value){{else}}{{placeholder}}{{/if}}" t
                      placeholder="{{placeholder}}"
                      class="form-control"
                      style="{{style}}"
                      data-constraints="{{id}}.validation('{{validation}}')"

                      {{#each dataatts}}
                      data-{{label}}="{{{value}}}"
                      {{/each}}
                      />

                    {{> _addDeleteButtons}}

                    {{> _showErrors}}
                {{#if render}}
                #end
                {{/if}}
            </div>

            {{#if render}}
            #end
            {{/if}}


            {{#if helptext}}
            <p class="help-block">{{helptext}}</p>
            {{/if}}

    {{#if label}}

        {{> _labelBottom}}
            </div> {{!-- Close component div --}}
        {{> _labelRight}}

    {{else}}
        </div> {{!-- Close component div --}}
    {{/if}}
```

```
<!-- END Text input-->

{{> _renderedClose}}
```

Model properties details

Property	Mandatory	Default	Note
fields	YES	{}	A map containing configuraiton options. The options names and values are used to *build* the actual widget's model
title	YES		The widget's title as displayed in the left sidebar of the FormBuilder
help	NO		The help message displayed in in the Form Builder configuration panel, as well as on the FormBuilder's left sidebar
order	NO		A list containing the widget's configuration's options names in the order in which they should be displayed in the configuration panel
jsdependencies	NO		List of static javascript resources the widget depends on
cssdependencies	NO		List of static CSS resources the widget depends on
nonRendableWidgets	NO	false	if true the widget can be resized (if true columns field is automatically injected among the widget's model fields list) (default: true)
columns	NO	true	The help message displayed in in the Form Builder configuration panel
binding	NO	true	if true the widget can be bound to an attribute of the Form Template
style	NO	true	if true the field *Custom Style* is automatically injected among the widget's model fields list.
validation	NO	true	True if the widget support validation (default:true)
readonly	NO	true	if true the field *Read Only* is automatically injected among the widget's model fields list.
container	NO	false	if true the widget will act as a container. The final view source code for all the widgets that are, in the Form Builder's working area, between the container opening and closing widget will result wrapped by the source code generated by the widget itself. When dropped in the Form Builder's main working area the corresponding closing widget will be automatically created and bound to it. The closing widget shall be named after the opening widget and suffixed with **_closed**.
rendered	NO	true	True if the designer should be able to specify a condition under which the widget will be displayed ("Show if" configuration option)

{{#if render}} expression in Widgets templates

As previously discussed, widget templates are mainly used to generate the VVTD, however they are also used to generate the HTML code that represents the widget in the FormBuilder workspace. When the Widget template is evaluated to generate the HTML for the FormBuilder workspace, an additional "render" property is injected into the widget model, so the designer has the possibility to filter elements that should not be rendered in static HTML. (e.g. any Velocity *(https://velocity.apache.org/)* expression).

Designers can modify widgets' models properties using the Form Builder widgets configuration panel. Any a widget's model modification triggers the immediate re-evaluation of the widget's template resulting into an update of the source code.

Static Resources Management¶

Beautiful WebForms widgest might depend on static resources (Javascript and CSS files). These dependencies are defined in the widget's model through the properties **jsdependencies** and **cssdependencies**.

The definition of a static-resource dependency is represented by a list of three elements:

- the reltaive [2] path to the static resource file
- the version of the resource to load (a string formatted as "Major.Minor.Revision")
- an optional list of dependency definitions for static resources this library depends on

E.g.

```
["v2/css/select2/select2-bootstrap","3.5.4", [ ["v2/css/select2/select2","3.5.4"] ] ]
```

When a form is rendered the framework computes the list of all the static resources required by the associated view's widgets. The list is optimized to avoid repetitions and to respect the proper loading order. The final list of static dependencies is then automatically injected by the framework in two ViewParams *(/working/bwebforms/views/#viewparams)* variables:`am_CssViewDependecies` and `am_JsViewDependecies`.

Beautiful webForms View Templates utilize the aformentioned variables to render the HTML code required to load the associated static files.

Two Velocity macros have been designed to handle this task:

```
#macro( bwfJsResources $resList $blackList )
#macro( bwfCssResources $resList $blackList)
```

These macros combine the contents of the variables `am_CssViewDependecies` and `am_JsViewDependecies` with the list of dependencies specified as macro arguments (which are typically dependencies specific to View Template *(/working/bwebforms/views/#beautiful-webforms-view-templates)*) to calculate the final list of static resources that must be loaded (producing at the same time the relevant HTML code).

$blacklist resources not to be loaded

It is sometimes desirable that the static resources that need to be loaded to satisfy a widget's dependency are not actually loaded, for example because they have been replaced by other resources already loaded by the View Template *(/working/bwebforms/views/#beautiful-webforms-view-templates)*, in these cases it is possible to pass to the above mentioned macros an additional optional list of resources not to be loaded.

E.g.

```
#bwfCssResources([
    ['v2/css/am/am_form', "2.0.0"]
    ,['v2/css/font-awesome.min', "0.0.0"]
    ,['v2/css/metro-bootstrap.min', "0.0.0"]
    ,['v2/css/am/am_gridTable', "2.0.0"]
    ,["v2/css/select2/select2-bootstrap","3.5.4",
        [
            ["v2/css/select2/select2","3.5.4"]
        ]
    ]
],
[ ["v2/css/bootstrap.min","3.3.6"]])
```

There are situations in which it is necessary to load multiple views dependecies when a WebForm is rendered:

- It is necessary whenever the WebForms can programmatically swithc view (e.g. a Webform organized in tabs);
- It is necessary whenever the WebForm's View makes use of SubViews widgets;

In these cases it is possibile to use the Content Script `forms.addResourceDependencies` API in the view OnLoad *(/working/bwebforms/views/#custom-logic-execution-hooks-cleh)* CLEH Script to force the framework to also load static resources dependencies from other Views.

The above mentioned API accepts three parameters: `forms.addResourceDependencies(boolean loadJS, boolean loadCSS, String[] viewNames)`

- A boolean flag indicating if Javascript resources should be loaded;
- A boolean flag indicating if CSS resources should be loaded;
- An optional list of Views from where to load dependecies from, if not specified resources will be loaded for all the Views associated with the parent Form Template object;

View Names

Prior to Module Suite version 2.7 *(/releasenotes/2_0_0/)* Views names had to be specified in single quotes.

E.g.

```
forms.addResourceDependencies(true, true, "'View2'", "'View3'")
```

Starting with Module Suite version 2.7 *(/releasenotes/2_0_0/)* Views names have be specified without quotes.

E.g.

```
forms.addResourceDependencies(true, true, "View2", "View3")
```

Performances-tips: Always load the minimum amout of resources necessary

When a Beautiful WebForm View is created the framework automatically injects in the OnLoad *(/working/bwebforms/views/#custom-logic-execution-hooks-cleh)* CLEH Script the code required to load static resource

dependecies from all the other views beloging to the same parent Form Template object. This code works well and has no impact on the performance of WebForm rendering, in most cases because Form Templates usually have very few associated views. However , there are situations in which this behaviour is not desirable (e.g. the Form Template contains many indipendent Views, the Form Template contains *non active* views etc..). loading static resource dependecies from other Views when unnecessary could be expensive and even lead to hardly detectable errors (e.g. a view in the template uses a different version of the widget library).
It's highly recommended, if your Form Template contains more than one view, to review the code automatically injected by the framework and modify it by passing to the `forms.addResourceDependencies` API (line 3) the list of Views from which it is actually necessary to load the resources.

```
1
2  form.viewParams.ajaxEnabled=true
3  if(form.viewParams.ajaxEnabled && !form.viewParams.isResourcesInit){
4      forms.addResourceDependencies( form, true, true)
5      form.viewParams.isResourcesInit = true
6  }
7  if (form.isFirstLoad()){
8    //Code to be executed on first load only
9    // es. form.myField.value = 'my value'
10
11 }
12 else{
13
   }
```

Widgets libraries¶

A **Widgets library** is defined as an extensible set of Widgets that can be used through the drag & drop visual editor (FormBuilder). To simplify the navigation, the widgets are arranged in families of objects having similar functionalities. Widgets within the same library use the same initialization mechanism, as far as the JavaScript and CSS frameworks are concerned. Whenever it is necessary or convenient to introduce breaking changes, in the way in which the widgets are defined or in the way in which the widgets are managed, a new library is released.

No need to update

Beautiful WebForms is always shipped with a copy of all still supported previous libraries. When a new library is issued, customers are not required to immediatelly upgrade their views to it. They are free to keep working with previous widget libraries.

Do not mix libraries

Given the nature of the differences between different libraries it is of highly recommended not to use widgets on different libraries in the same view. Mixing widgets from different libraries can lead to unpredictable results or errors.

Widget Library V1¶

This is the first version of the widget library shipped with the first version of Module Suite. This widget library has been retired and is no longer supported since Module Suite version 2.6 (/

releasenotes/2_6_0/). View Templates designed to work with library V1 are not compatible with any other library. Do not use other libraries' widgets with these View Templates.

Widget Library V2¶

This version of the widget library was first introduced with Module Suite 2.0 *(/releasenotes/2_0_0/)* and is still fully supported. This library is the first using the concept of static resources management. View templates leveraging this library loads their static resource dependencies through standard HTML tags `<link>` and `<script>`. The actual HTML code required to load resources is produced by the two Velocity macros (*bwfCssResources* and *bwfJsResources*) mentioned in the static resources management paragraph. View Templates designed to work with library V2 are not compatible with any other library. Do not use other libraries' widgets with these View Templates.

Widgets of library V2 have two additional model properties: *jsdependencies* and *cssdependencies*, they represent the list of static javascript and css resources the widget depends on:

The definition of a static-resource dependency is represented by a list of three elements:

- the reltaive [2] path to the static resource file
- the version of the resource to load (a string formatted as "Major.Minor.Revision")
- an optional list of dependency definitions for static resources this library depends on

E.g.

```
...
jsdependencies:[ ["v2/css/select2/select2-bootstrap","3.5.4", [ ["v2/css/select2/select2","3.5.4"] ]
...
```

Widget Library V3¶

This version of the widget library was first introduced with Module Suite 2.4 *(/releasenotes/2_4_0/)* and is still fully supported. This library revised the concept of static resources management. View templates leveraging this library loads their static resource dependencies through standard HTML tags as far as CSS resources are concerned and a JavaScript file and module loader Require JS *(https://requirejs.org/)* for Javascript resources. The actual HTML code required to load CSS resources is produced by the the Velocity macro (*bwfCssResources*) mentioned in the static resources management paragraph. View Templates designed to work with library V3 are not compatible with any other library. Do not use other libraries' widgets with these View Templates.

Widgets of library V3 have two additional model properties: *jsdependencies* and *cssdependencies*, they represent the list of static javascript and css resources the widget depends on:

CSS dependeciesJS dependecies

The definition of a static-resource CSS dependency is represented by a list of three elements:

- the reltaive [2] path to the static resource file
- the version of the resource to load (a string formatted as "Major.Minor.Revision")
- an optional list of dependency definitions for static resources this library depends on

E.g.

```
...
"cssdependencies":[
  ["v3/js/handsontable/handsontable.full","4.0.0", [["v3/js/handsontable/pikaday","1.4.0"]]]
  ,["v3/css/select2/select2","3.5.4"]
]
...
```

The definition of a static-resource JS dependency is represented by a list of three elements:

- the reltaive [2] path to the static Javascript bundle containing the modules to be loaded
- the version of above mentioned bundle (a string formatted as "Major.Minor.Revision")
- the list of module that are part of the bundle (modules are defined by a list made of their name and version)

```
...
"jsdependencies":[
    ["v3/js/handsontable/am_init","1.0.0",[["Handsontable","4.0.0"], ["pikaday","1.4.0"], ["numbro",
]
...
```

Widget Library V4¶

This version of the widget library was first introduced with Module Suite 2.6 *(/releasenotes/2_6_0/)* and is still fully supported. This library it's an evolution of the previous iteration (library V3) which significantly increases the compatibility with standard Smart View UI. View templates leveraging this library loads their static resource dependencies through standard HTML tags as far as CSS resources are concerned and a JavaScript file and module loader Require JS *(https://requirejs.org/)* for Javascript resources, which is the same AMD library used by native Content Server Smart View framework. The actual HTML code required to load CSS resources is produced by the the Velocity macro (*bwfCssResources*) mentioned in the static resources management paragraph. View Templates designed to work with library V4 are not compatible with any other library. Do not use other libraries' widgets with these View Templates.

Widgets of library V4 have two additional model properties: *jsdependencies* and *cssdependencies*, they represent the list of static javascript and css resources the widget depends on:

CSS dependeciesJS dependecies

The definition of a static-resource CSS dependency is represented by a list of three elements:

- the reltaive [3] path to the static resource file
- the version of the resource to load (a string formatted as "Major.Minor.Revision")
- an optional list of dependency definitions for static resources this library depends on

E.g.

```
...
"cssdependencies":[
    ["amui/handsontable.full","4.0.0", [["amui/pikaday","1.4.0"]]]
    ,["amui/select2/select2","3.5.4"]
]
...
```

The definition of a static-resource JS dependency is represented by a list of three elements:

- the name of the Javascript bundle containing the modules to be loaded, the bundles and the names of the modules no longer contain references to the name of the library version
- the version of above mentioned bundle (a string formatted as "Major.Minor.Revision")
- the list of module that are part of the bundle (modules are defined by a list made of their name and version)

```
...
"jsdependencies":[
    ["bwf/handsontable/am_init","1.0.0"]
]
...
```

---

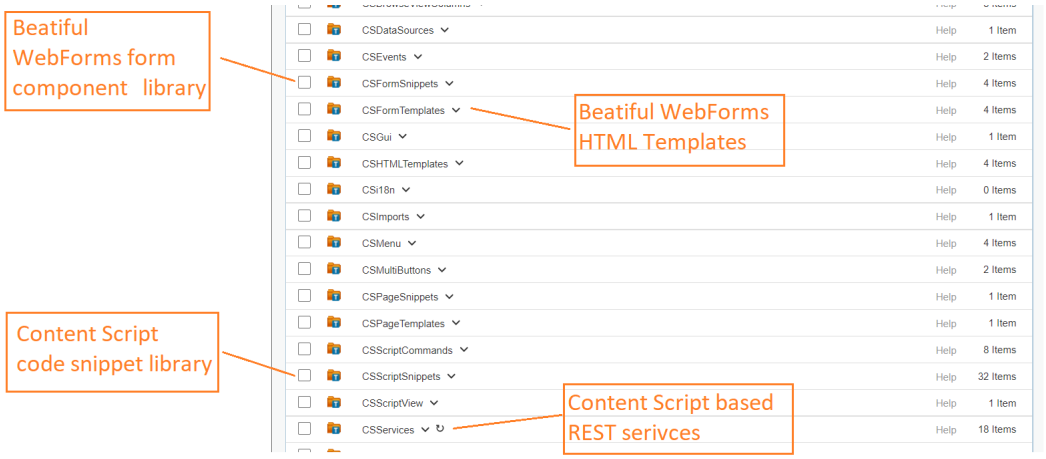
1. FormBuilder acts as a Model View Controller framework with respect to BWF Widgets
2. Paths are relative to the folder **/support/ansbwebform/lib**
3. Paths are relative to the folder **/support/ansbwebform/lib/v4**, paths are defined in the View Template through Velocity expressions

Extending BWF

Content Script Volume¶

As for Content Script, Beautiful WebForms makes use of the Content Script Volume to store a set of objects necessary for the correct operation of the framework. These object are stored in specific containers, which will be covered on the following sections:

- CSFormTemplates
- CSFormSnippets
- CSServices
- CSScriptSnippets



CSServices¶

The **CSServices** *(/working/contentscript/rest/)* container is dedicated to Content Scripts that should be accessible as REST services, and has been covered in the previous sections.

Content Script REST services are somehow related to Beautiful WebForms in that some components used to build forms (essentially, the ones with AJAX capabilities) make use of these services to work correctly.

An example is the **getuserbyname** REST service, which backs the user selection components available in the form builder.

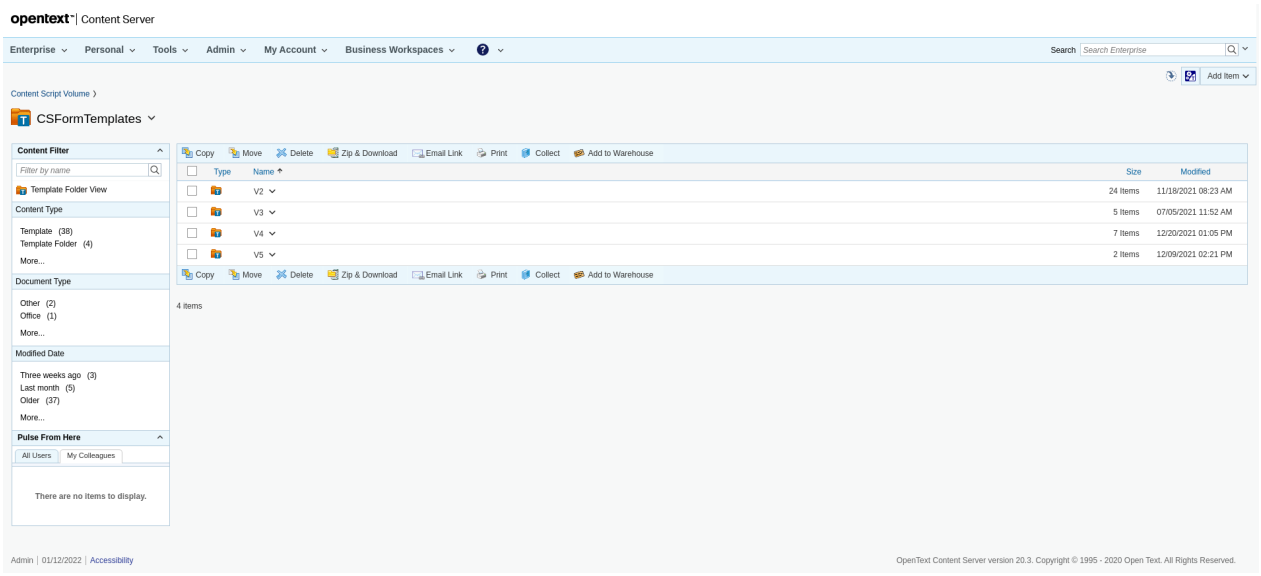
CSFormTemplates¶

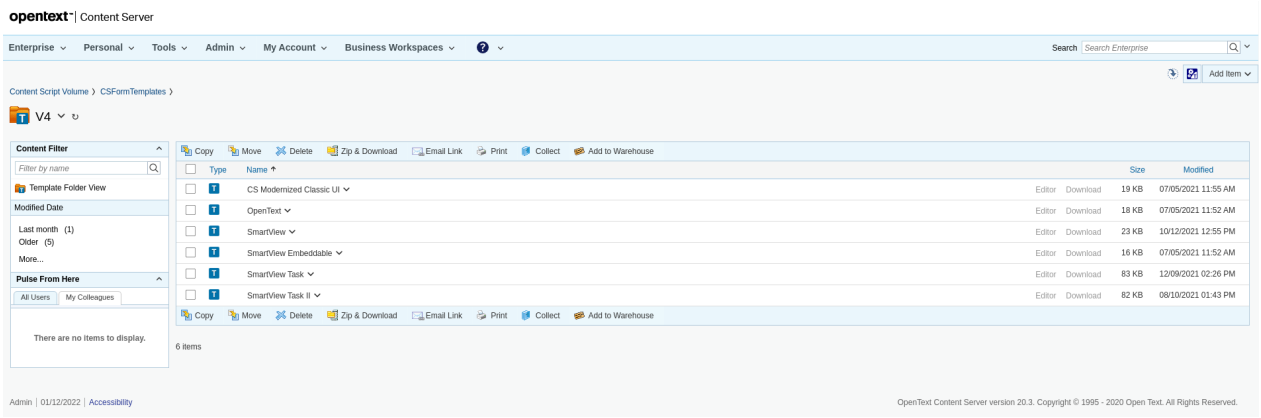
The **CSFormTemplates** container is dedicated to HTML templates associated to **Beautiful WebForms Views**.

The templates are essentially **Velocity** HTML templates. A placeholder expression indicating where the actual Form Fieldset should be placed, this should usually be present in all Beautiful WebForms Templates.

Beatiful WebForms Templates are grouped by the library version:

- Content Script Volume
  - CSFormTemplates
    - V2
    - V3
    - V4
      - *<custom template A>*
      - *<custom template B>*





New templates added to a library version folder will automatically be available in the **template selection** dropdown menu accessible from the Beautiful WebForms Views Specific Properties tab.

CSFormSnippets¶

The **CSFormSnippets** container is dedicated to the libraries of **components** that are available to build Beautiful WebForms views.

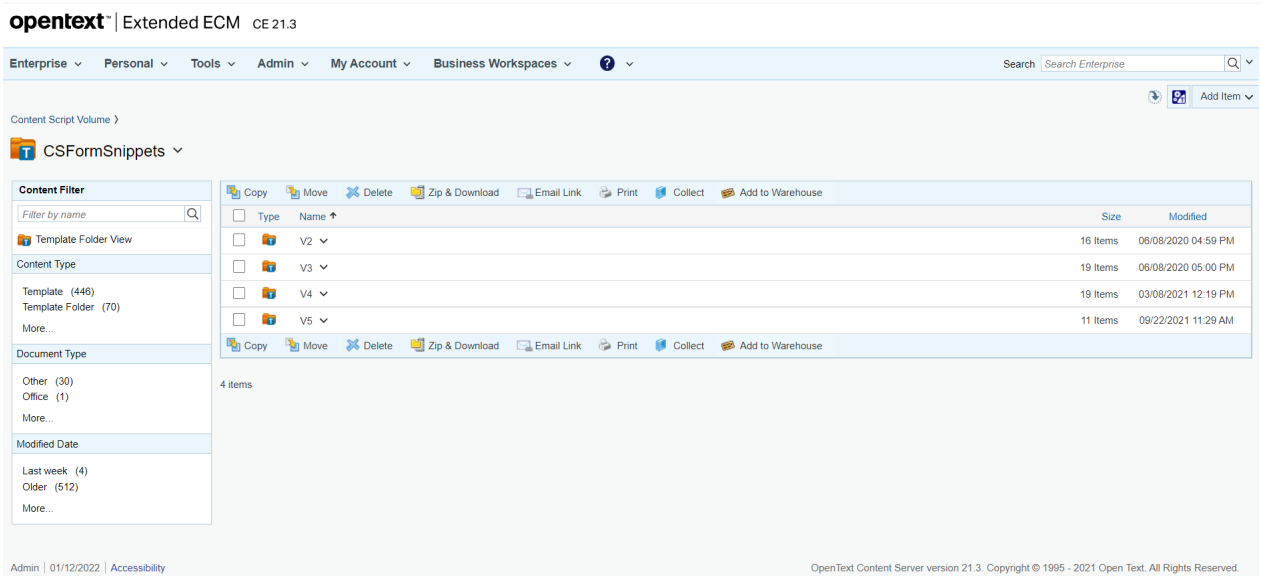
The CSFormSnippets container is organized on two levels: the first level is a container and identifies the Component Family, while at the second level there are the actual components.

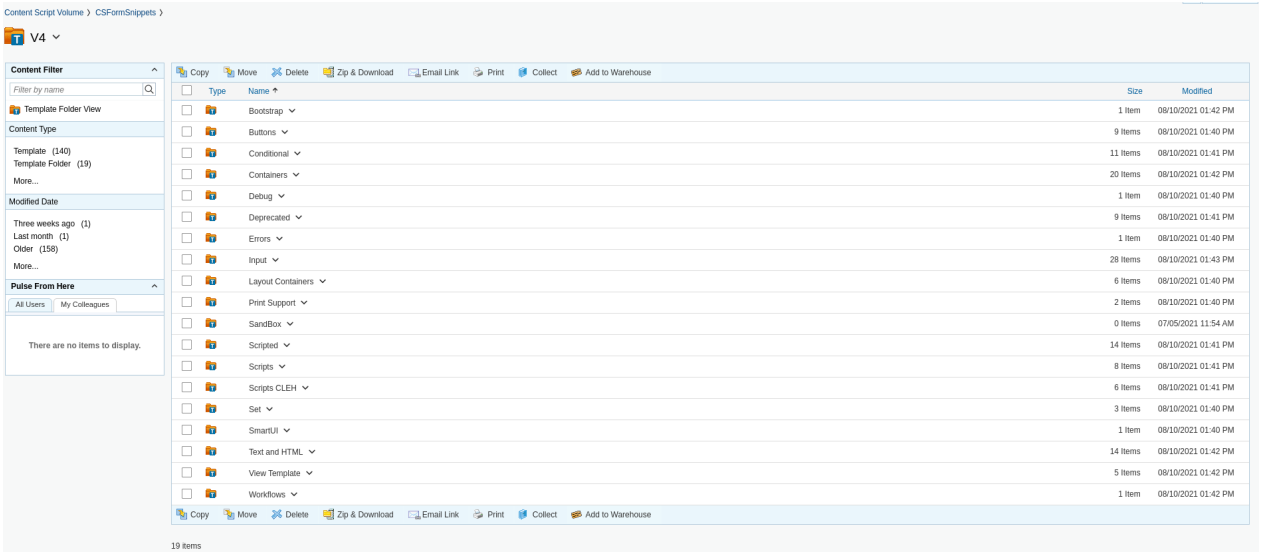
The Beatiful WebForms Snippets are stored in a two levels folders hierarchy: the first level is a library version container, while the second level is a container that identifies the Component Family.

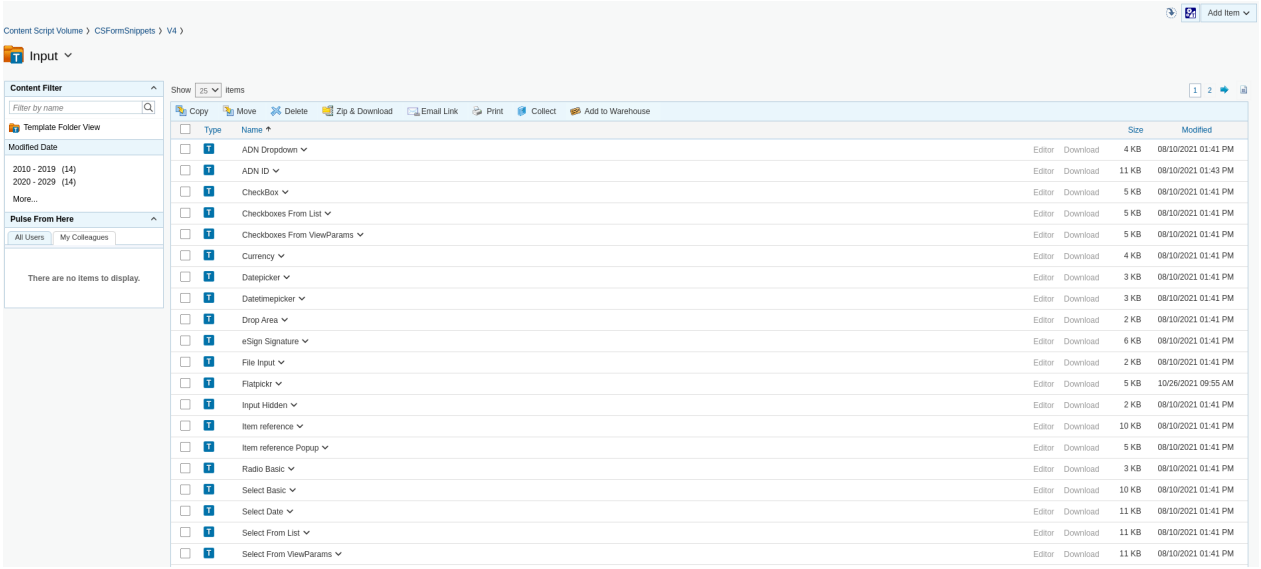
- Content Script Volume
  - CSFormSnippets
    - V2
    - V3
    - V4 - *library version level*
      - Buttons - *Component Family level*
      - Input
        - CheckBox
        - Datepicker
        - Text input
        - *<custom component A>*

- Set

New component families and components created in this container will automatically be available to the developer in the Beautiful WebForms Form Builder tool.







Embed into Smart View¶

Why?¶

The main purpose of embedding BWF views into Smart View's tiles is to leverage the BWF framework as a primary input mechanism for your next EIM applications. Integrating BWF into Smart View wont just enable you to collect and validate user's input but also to perform complex actions and surface the most relevant business information in highly interactive dashboards.

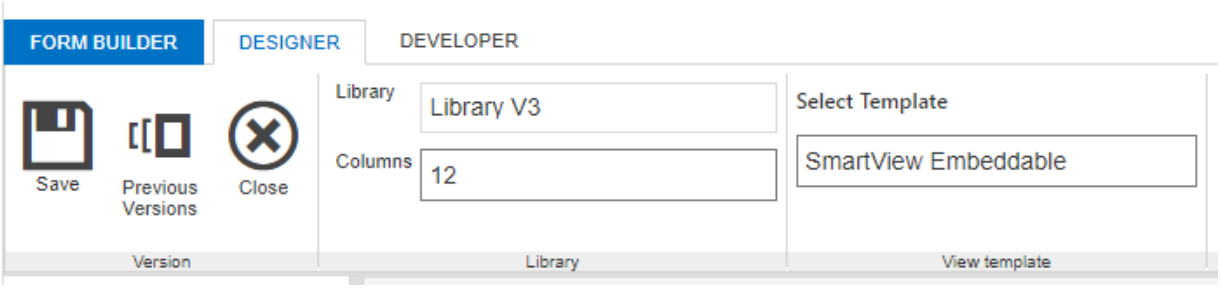
Create an embeddable WebForms¶

Creating an *embeddable webforms* is not different from creating any other webform on the system. The steps are:

- Create a **Form Template** object
- Create a **Beautiful WebForm View** view associated to the **Form Template** created in the previous step
- Using the Beautiful WebForms Form Builder define your form (structure and layout)

> **The embeddable view template**

What makes a Beautiful WebForms view embeddable into the Smart View is the usage of the **V3:SmartView Embeddable** view template



- Create a standard Content Server Form object and associate it to the previously created **Form Template** and **Beautiful WebForm View**

How to publish a Webform into a Smart View perspective¶

In order to publish a WebForm in a Smart View perspective's tile you need either:

ModuleSuite Smart Pages is installed

1. A Content Script object (for managing the server side initialization of the form)
2. An **AnswerModules ModuleSuite:Content Script Result** perspective tile, configured to use the above script as datasource

or

ModuleSuite Smart Pages is not installed

1. A Content Script object (to mange the server side initialization of the form)
2. A WebReport to encapsulate the above script execution
3. An **Content Intelligence:HTML WebReport** perspective tile, configured to use the above script as Webreport as datasource

ModuleSuite Smart Pages is installed¶

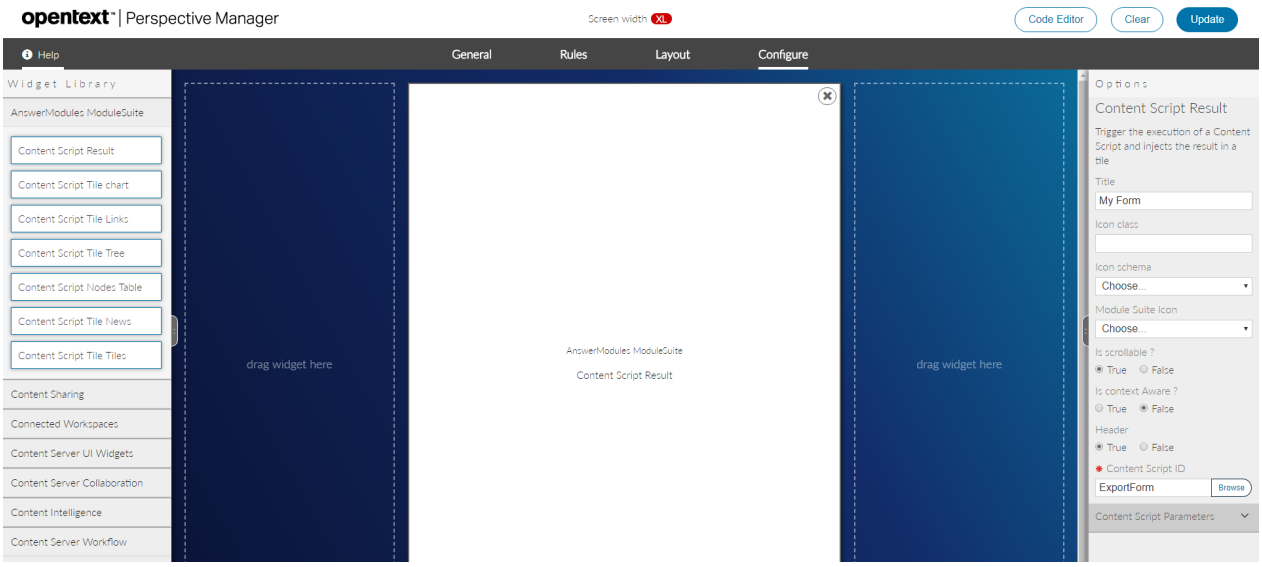
If the ModuleSuite Smart Pages is installed on your system you will be able to leverage the tight integration between ModuleSuite and the OTCS Smart View in order to add WebForms in perspective's tiles.

In this case the minimum Content Script required for managing the server side initialization of the form will be:

```
def formNode = docman.getNodeByPath("Path:To:Your:Form")
form         = formNode.getFormInfo()
```

```
view         = formNode.view
form.viewParams.uiParentID = params.uiParentID //The perspective current space

json([
    output:view.renderView(binding, form),
            widgetConfig:[
                reloadCommands:["someCommand"],
                tileContentClasses:"am-whitebckg",
                tileLayoutClasses:"am-whitebckg"
            ]
        ]
)
```

The configuration of the associated **AnswerModules ModuleSuite:Content Script Result** will be as simple as:



ModuleSuite Smart Pages is not installed¶

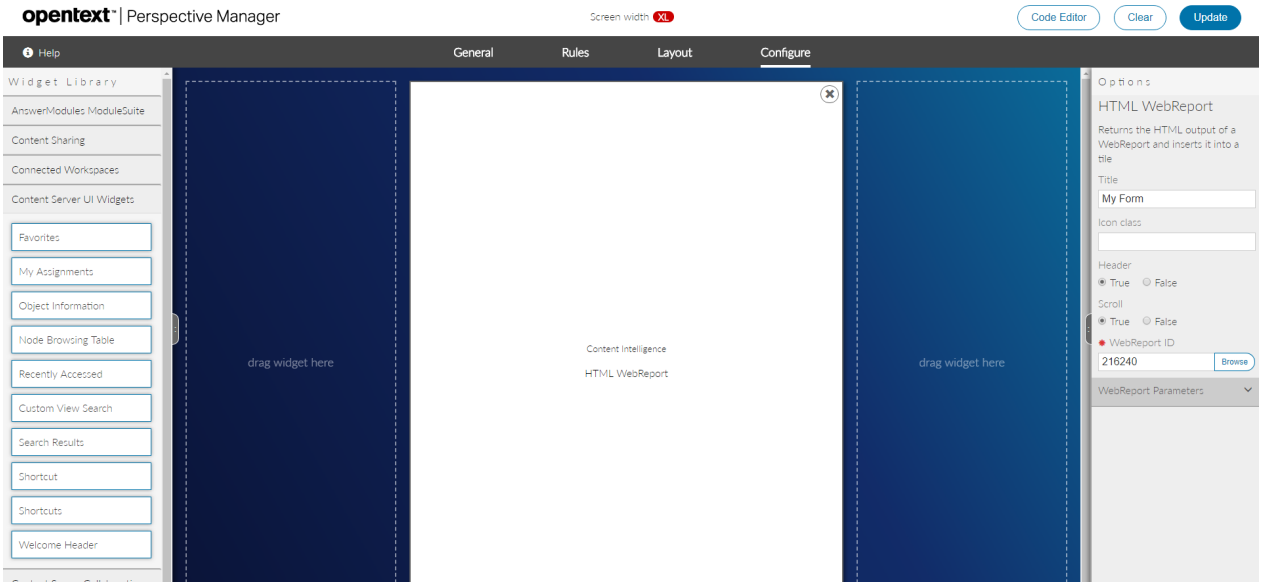
If the ModuleSuite Smart Pages is not installed on your system you will not be able to leverage the tight integration between ModuleSuite and the OTCS Smart View (i.e. adding WebForms in perspective's tiles). As an alternative to Smart Pages you can leverage **WebReport** in order to encapsulate a Beautiful Webform in a Smart View Perspective. In this case what you have to do is to use the **RUNCS** sub-tag to trigger the execution of a properly configured Content Script. You can refer to the example below for a reference:

```
gui.gui = false
def formNode = docman.getNodeByPath("Path:To:Your:Form")
form         = formNode.getFormInfo()
view         = formNode.view
out << view.renderView(binding, form)
```

The WebReport required to encapsulate the execution of the above script will be:

```
[LL_REPTAG_'123456' RUNCS /] [// Script ID
[LL_WEBREPORT_STARTROW /]
[LL_WEBREPORT_ENDROW /]
```

The configuration of the associated **Content Intelligence:HTML WebReport** will be as simple as:



Beautiful Webforms views updater¶

What is it?¶

The Beautiful Webforms View Updater (BWVU) is an utility designed to simplify and automate the process of upgrading a webform view designed with a previous version of Module Suite. Module Suite IDEs allows you to keep working with the views created using the widget library shipped with a previous version of Module Suite, nevertheless, in order to leverage the widgets introduced in a newer version of the widget's library an upgrade is required.

This tool aims to simplify the upgrade procedure.

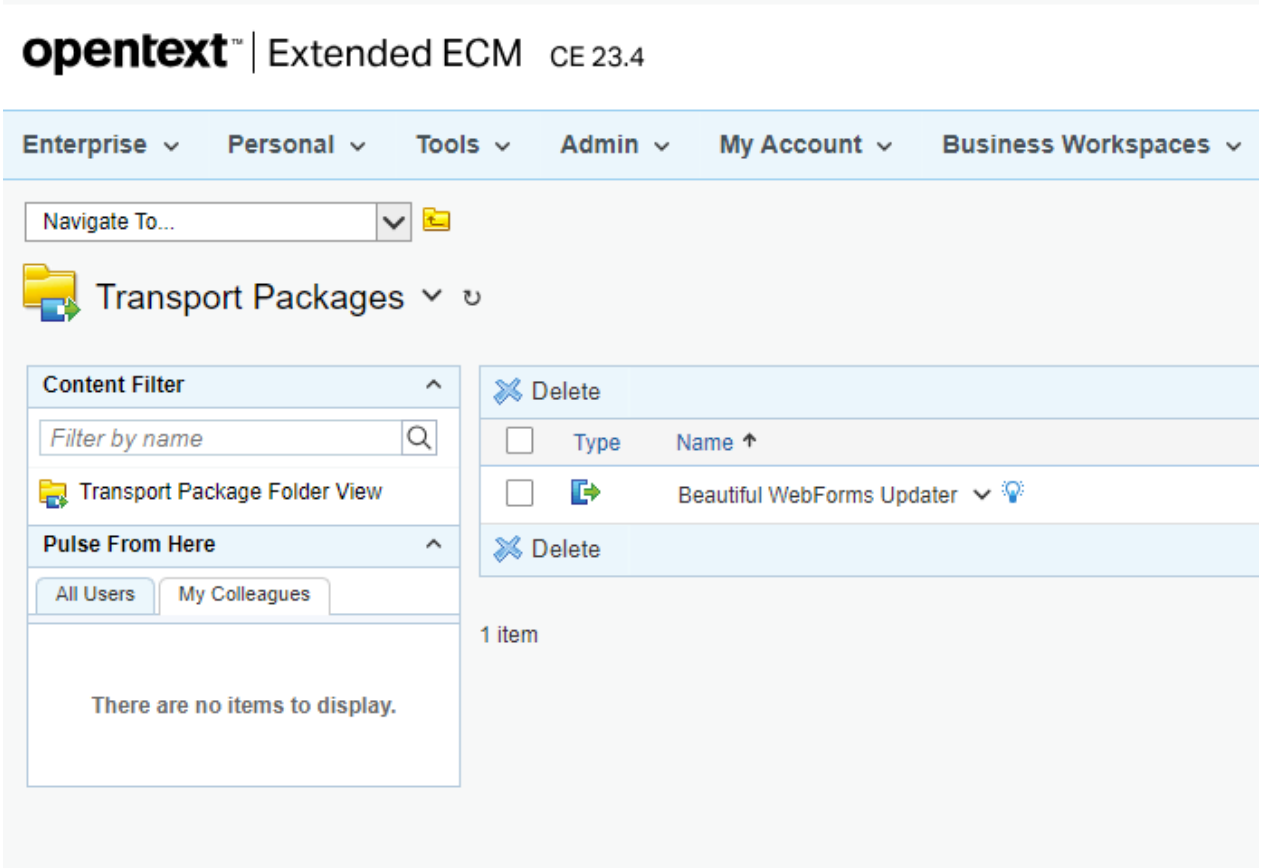
Installation¶

Prerequisites¶

Ensure you have administrative access to the OpenText Content Server to install the Beautiful WebForms Updater.

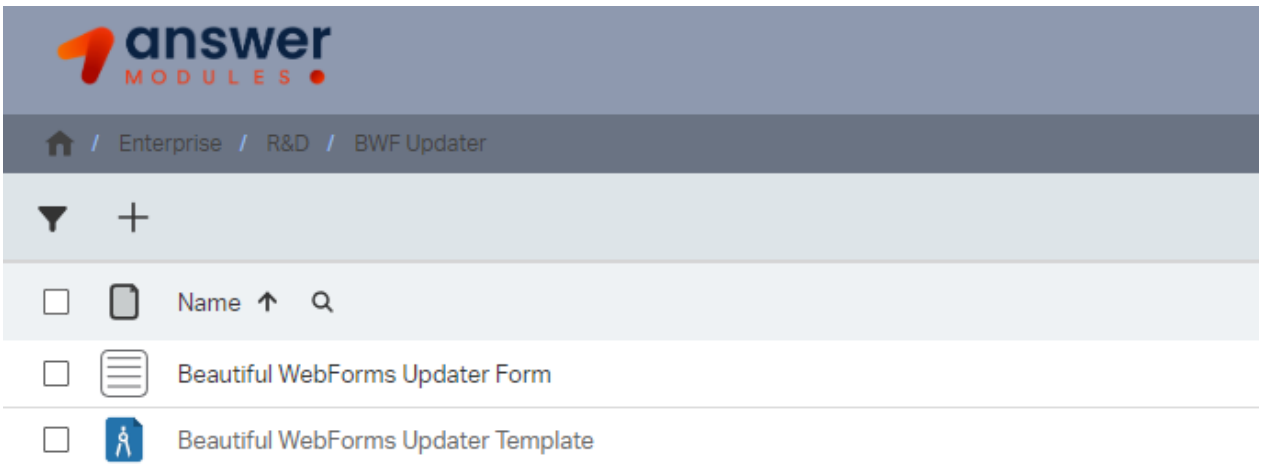
Installation Steps¶

1. The Beautiful WebForms Updater is distributed as a standard Warehouse transport package and can be downloaded from here.
2. For detailed steps on how to deploy a transport package, refer to the OpenText Content Server administration's guide.



Getting Started¶

Once installed, access the Beautiful WebForms Updater tool by clicking on the **Beautiful WebForms Updater Form** form.

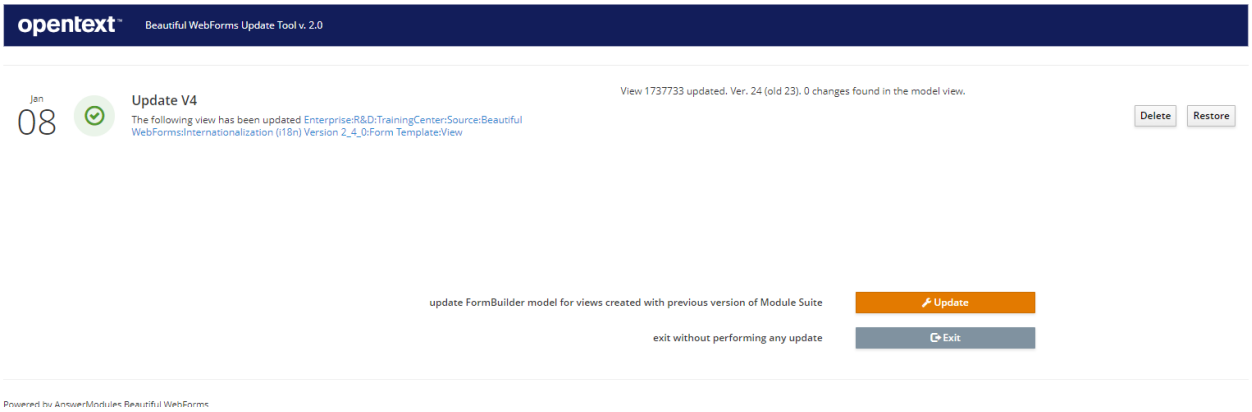


Main Dashboard¶

Upon opening the tool or after having performed an update operation, you are redirected to the main dashboard. This area provides an activity log and the ability to manage the updates performed.

Dashboard Features¶

- **Activity Log**: Lists all actions taken, along with timestamps.
- **View Details**: Includes the name of the view, the action taken, and the update or restoration details.
- **Action Buttons**: Offers "Delete" or "Restore" options for each log entry.



Navigating the Main Dashboard¶

- To update the log, navigate away and return to the dashboard.
- Remove log entries using the "Delete" button.
- Use the "Restore" button to revert any updates, if necessary.

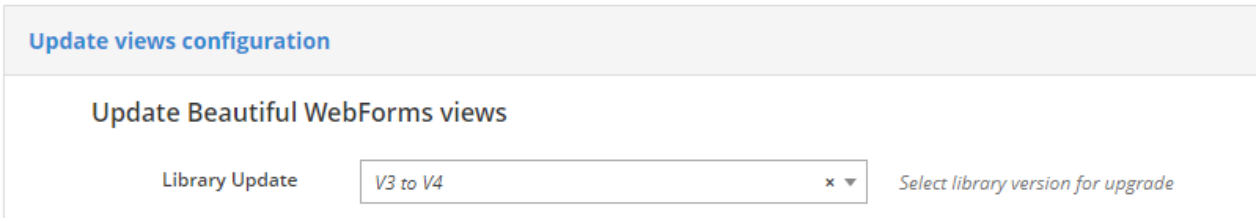
Update Views Configuration¶

To Proceed with the update your web forms click on the **Update** button from the **Main dashboard**, you will be redirect to the **Views Update Page**.

Views Update Page

Library Update¶

Select the library version you want to upgrade to or from. For example, upgrading from V2 to V3.

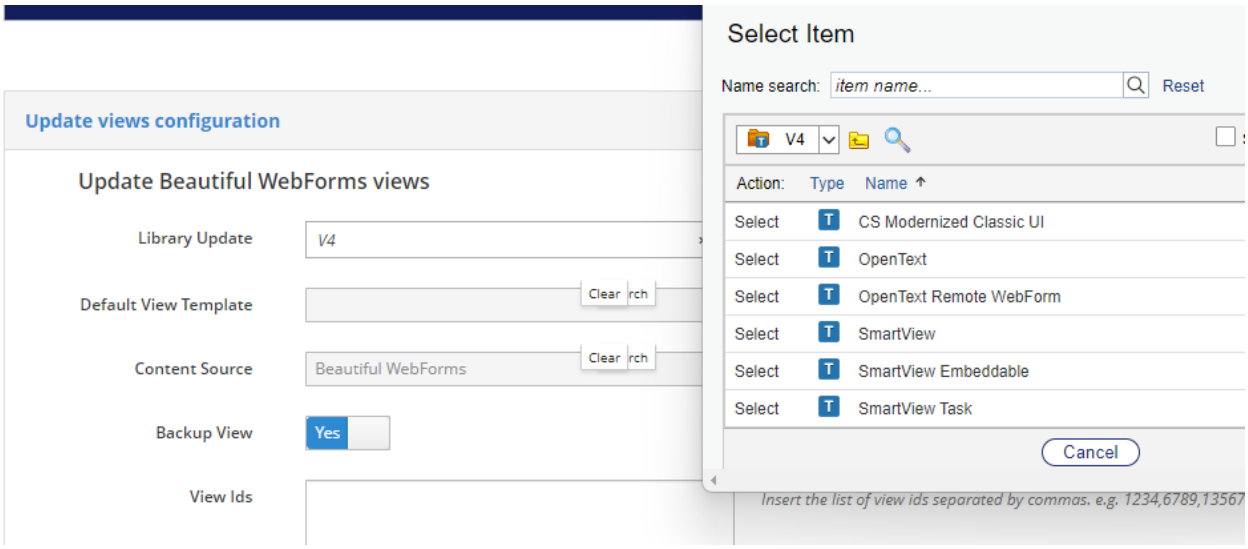


Import the widgets library before trying to update your views

The tool requires that whatever version of the library you wish to upgrade to be fully imported into the Content Script volume. The import can be managed using the Content Script Volume Import Tool

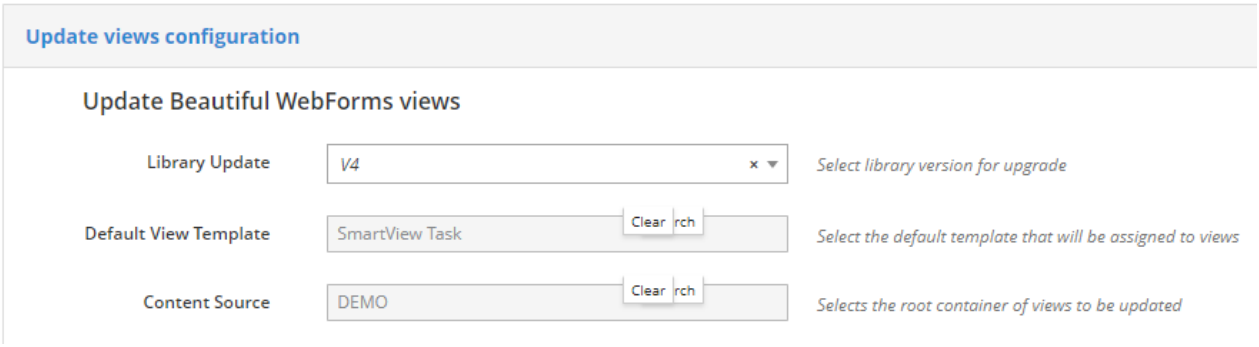
Default View Template¶

Choose a default template that will be associated with all views after the update.



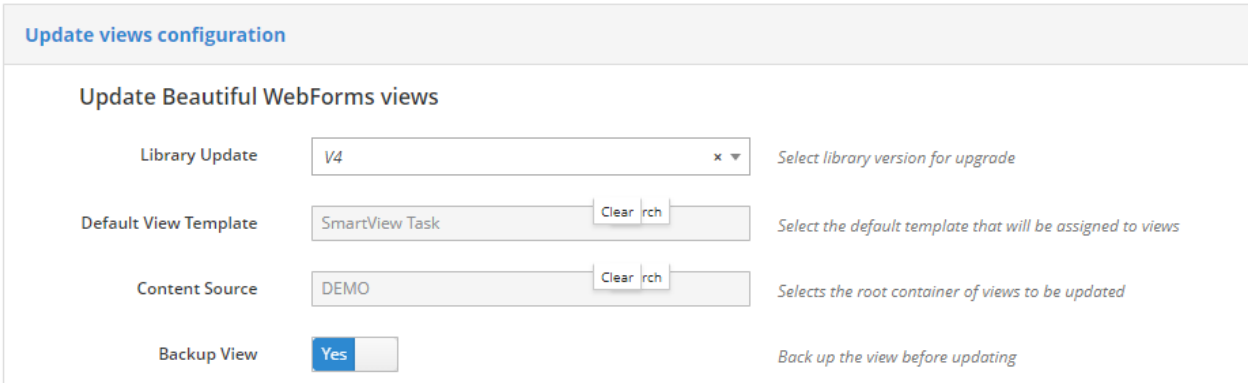
Content Source¶

Select the content source container of views to be updated.



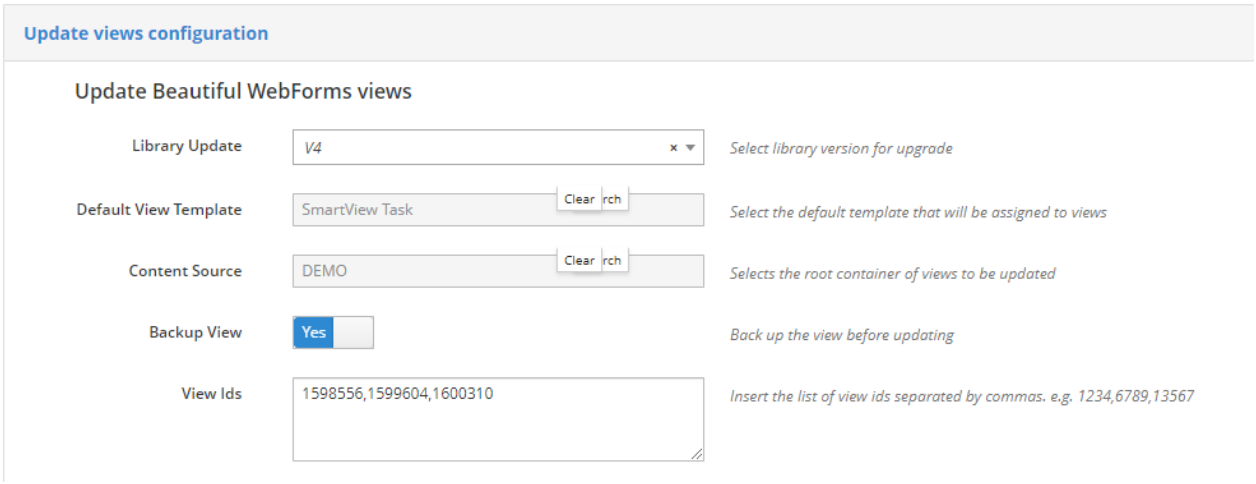
Backup View¶

Toggle this option to "Yes" to create an XML backup of the view before updating.



View Ids¶

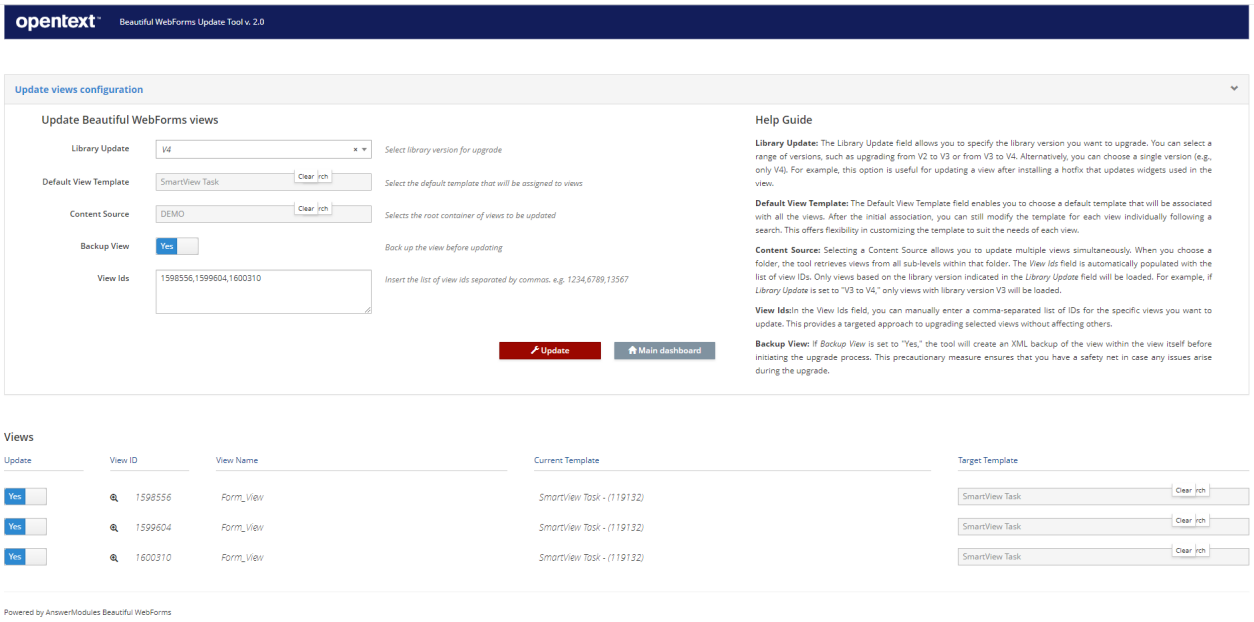
Enter the list of view IDs you wish to update, separated by commas.



Updating Views¶

1. Check the boxes next to the views you wish to update under the **Views** section.

2. Click the **Update** button to start the update process.



Help Guide¶

A Help Guide is available on the right-hand side of the page to assist you with the tool.

Troubleshooting¶

If you encounter any issues, refer to the Help Guide first, then contact our support team *(https://support.answermodules.com)*.

Conclusion¶

After following these steps, your views should be successfully updated with the new library version or template.

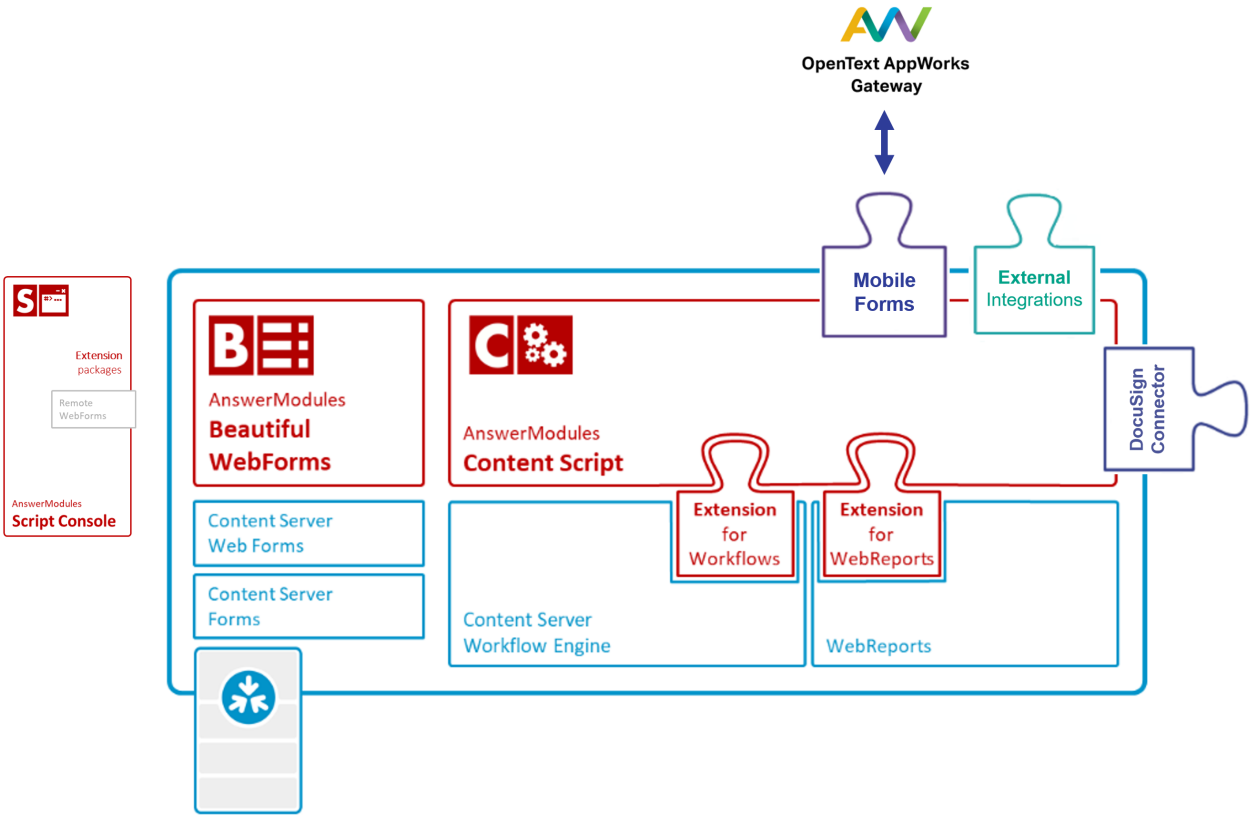
Extension: Mobile WebForms

Feature deprecated

This feature has been deprecated and removed from the product since version 2.9.

What is it?¶

AnswerModules' Mobile WebForms is both: - An add-on solution for CSP/xECM. - A functional extension for Module Suite (AnswerModules' core solution).



AnswerModules' Mobile WebForms consists of three macro components:

AppWorks Mobile Application¶

Every Mobile WebForms is transformed into an AppWorks application so that it can be it distributed to end-users' devices through the AppWorks Gateway. This approach guarantees a very high degree of flexibility in terms of controlling access to the mobile form as well as governing the mobile form's data security. By leveraging the AppWorks technology, a mobile form's lifecycle can be fully managed (versioning, fine-grain user distribution, etc..), support for specific devices may be pre-defined and if necessary saved data could be remotely deleted from a specific device.

Module Suite based extension for REST APIs¶

By extending the CSP/xECM REST APIs a dedicated endpoint for Mobile WebForms has been created. The endpoint can be easily extended or adapted in order to effectively open a potentially infinite number of use cases when it comes to how form data is utilized and persisted once its synchronized onto CSP/xECM. Some possible scenarios for how the form data can be utilized include: starting or updating a workflow, creating Connected Workspaces

programmatically, generating documents (PDF, Word, Excel, etc...), transmitting the data to another system (i.e.: CRM, ERP, etc...), and much more.

Mobile WebForms Application Builder¶

This component allows to create new AppWorks applications in a matter of minutes starting from an existing form. An intuitive wizard-like tool guides users in defining all the necessary elements to transform a simple WebForm into a Mobile WebForms. A preview of the process can be viewed at: https://youtu.be/xiBjPMAH-HU *(https://youtu.be/xiBjPMAH-HU)*

Mobile WebForms setup¶

Installing the Mobile WebForms application on your system is a straightforward procedure made of a few simple steps.

As administrator

The installation procedure must be performed using a user with administrative rights on the system (for example, the administrator user)

- Download the Mobile WebForms Installation Package. (You can download it from here)
- Extract the contents of the zip file to a temporary location.
- Copy the contents of the Mobile Components.zip in the <Content_Server_home> directory and then restart the Content Server services.
- Logon to the OpenText Content Server with an administrative account.
- Create a folder that will contain the installation package.
- Upload the mobileWebFormsXML.xml file, in the previously created folder.
- Create a Content Script in the same location for importing the package in the system. (please refer to the snippet below as a reference).

```
def source    = docman.getNodeByName(self.parent, "mobileWebFormsXML.xml")
def xmlFolder = docman.getNodeByName(self.parent, "Mobile WebForms")
if(!xmlFolder){
    xmlFolder = admin.importXml(self.parent, source.content.content)
}
redirect "${url}/open/${docman.getNodeByName(xmlFolder, 'Install').ID}?scriptInstall=${self.ID}"
```

The execution of the Content Script will generate a folder in the Enterprise Workspace named “MobileWebForms” and will generate the application’s contents in it.

Pre-requisites

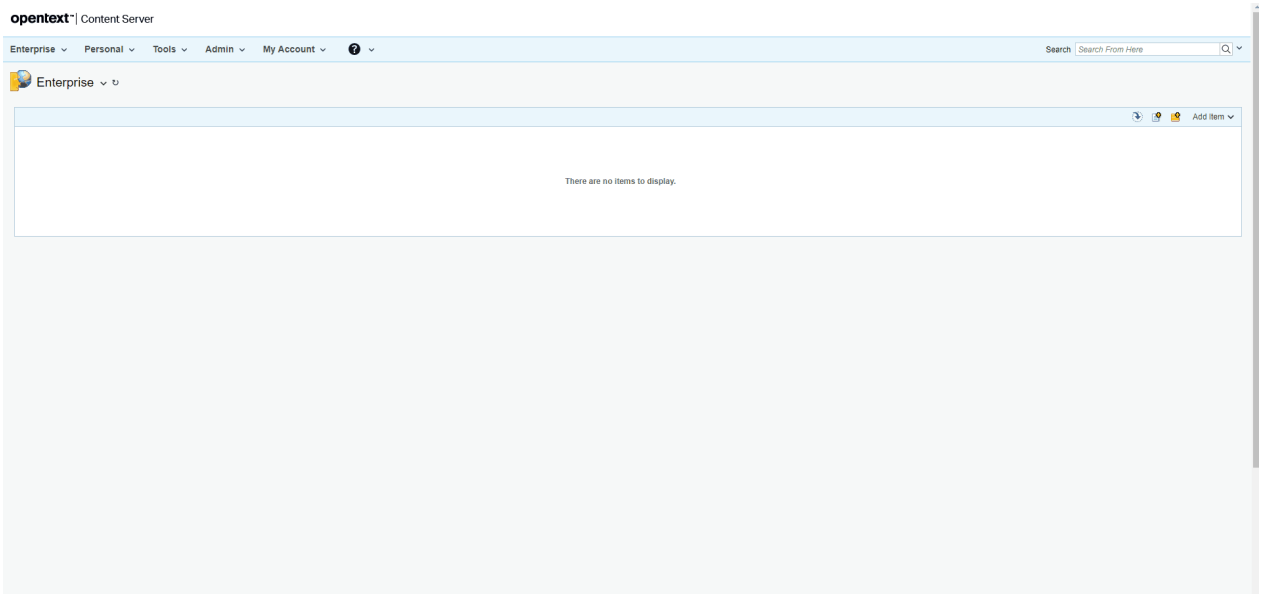
During the setup process the installer, will check if all the prerequisites are met. If the setup process notifies the need of a missing extension package, install the package before continuing.

To install an extension package you can refer to the following guide: http://developer.answermodules.com/manuals/current/installation/extpacks/ *(http://developer.answermodules.com/manuals/current/installation/extpacks/)*

In the case the requested extension is the AnswerModules' Cache Extension Package then after the installation some additional configuration will be needed.

To properly configure the AnswerModules' Cache Extension Package refer to the below guide:

https://support.answermodules.com/portal/kb/articles/content-script-extension-cache *(https://support.answermodules.com/portal/kb/articles/content-script-extension-cache)*



Using the tool¶

A Mobile WebForms application is composed of three main elements:

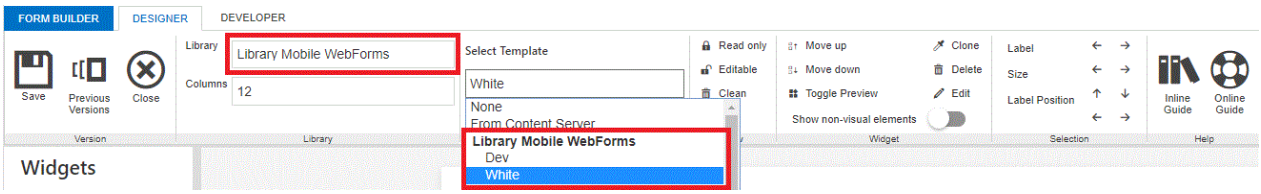
- A form for inserting the information.
- An end-point Content Script that will implement the logics to properly manage the data upon synchronization from the OpenText AppWorks Gateway application.
- An OpenText AppWorks Gateway application for distributing the application to the end users.

Creating the form¶

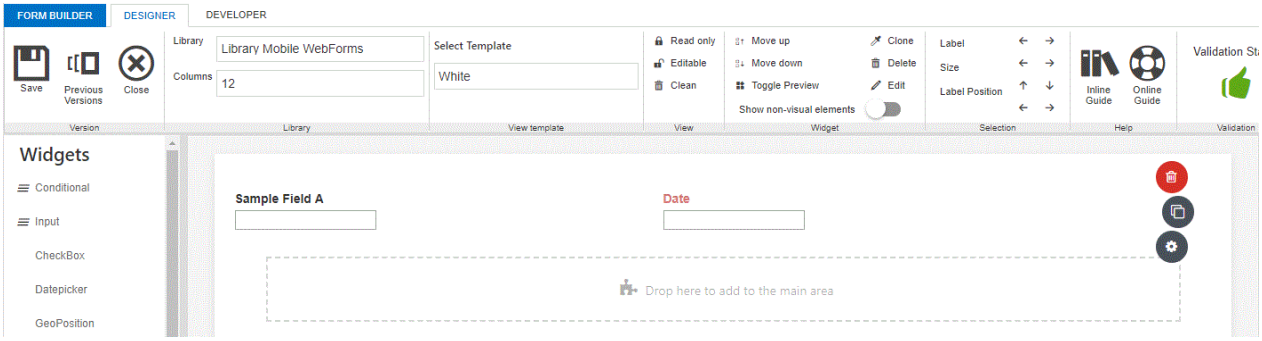
The first step is the creation of the form that will be utilized to gather information from the end users.

When editing the form's view with the Form Builder the widget library must be set to "Library Mobile WebForms". As for the template to use there are two options under the "Library Mobile WebForms" section:

- Dev: this template offers the possibility to verify the look & feel of the form without the need to deploy it on the OpenText AppWorks Gateway. This template should be only utilized during the development phase or for debugging purposes.
- White: this is template to be utilized when the application is ready to be deployed on the OpenText AppWorks Gateway.



When editing a form's view with the Form Builder, the form's view will be pre-populated with the widgets representing the elements inserted in the Form Template. A Mobile WebForms will need to be designed using specific widgets coming from the Mobile WebForms Library, to do so delete the self created widgets derived from the form template, verify that the Library Mobile WebForms is selected, save the form's view and refresh the page. Once the page has refreshed drag&drop the widgets from the left-hand side of the Form Builder to the form's view.



Implementing the Content Script end-point¶

When synchronizing the information back to Content Server, the Mobile WebForms application will make a call to a Content Script.

For a detailed explanation on using AnswerModules' Content Scripts please refer to the following guide: http://developer.answermodules.com/manuals/current/working/contentscript/otcsobj/ *(http://developer.answermodules.com/manuals/current/working/contentscript/otcsobj/)*

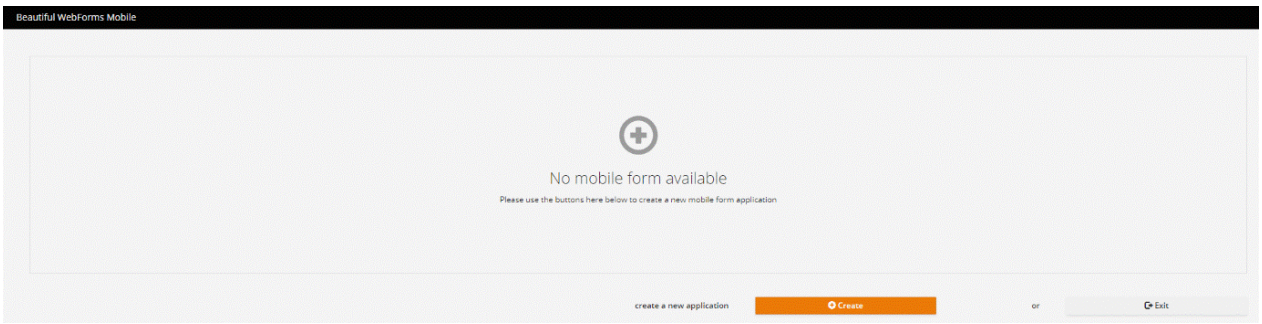
The Content Script must reside inside the CSServices folder within the Content Script Volume. The script must contain all the business logic needed to properly manage the information that is being synchronized from the OpenText Gateway application. The installation process will

create a default end-point called "mobileWebForms" please refer to it as a reference implementation.

Building the OpenText AppWorks Gateway Application¶

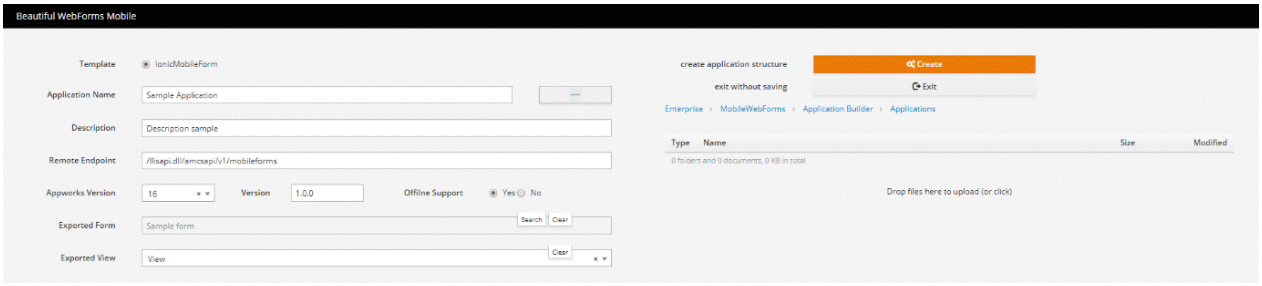
To deploy the application on the OpenText AppWorks Gateway it will be necessary to prepare a deployable package compliant with the OpenText AppWorks Gateway. The preparation of the up-said package can be done via the Mobile WebForms application by opening the form “Registered Applications”. The form can be found under Enterprise\MobileWebForms\Application Builder\Builder

Once opened, the form will show the list of registered application. New applications can be created by clicking on the "Create" button at the bottom of the page.

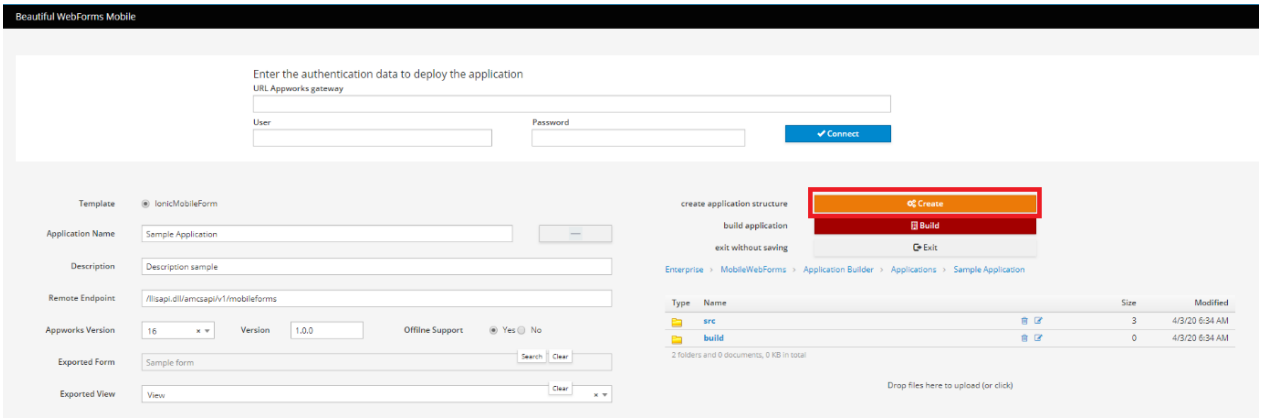


Clicking on the “Create” button will prompt the user for the application's details.

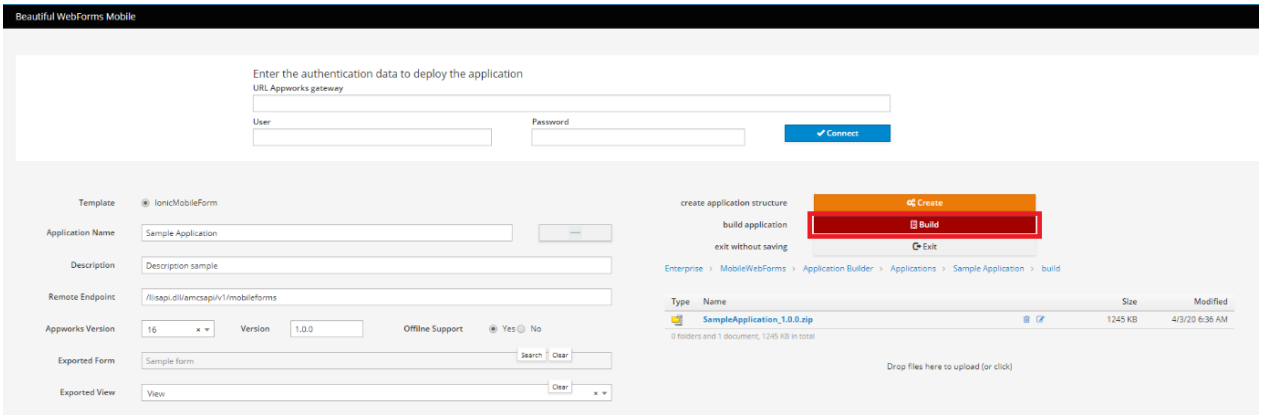
- Application name
- An icon for the application (to be shown as the application's icon on the mobile device)
- A description (to be set as the application's description on the mobile device)
- The remote end-point script name (called when synchronizing the form's data)
- The Appworks Gateway version
- The application’s version (When updating the application the version number must be increased)
- The related form
- The specific view to be used



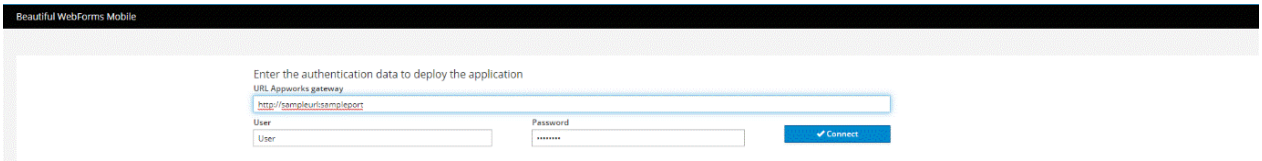
Clicking the “Create” button will automatically create an appropriate folder structure containing all the application's required objects



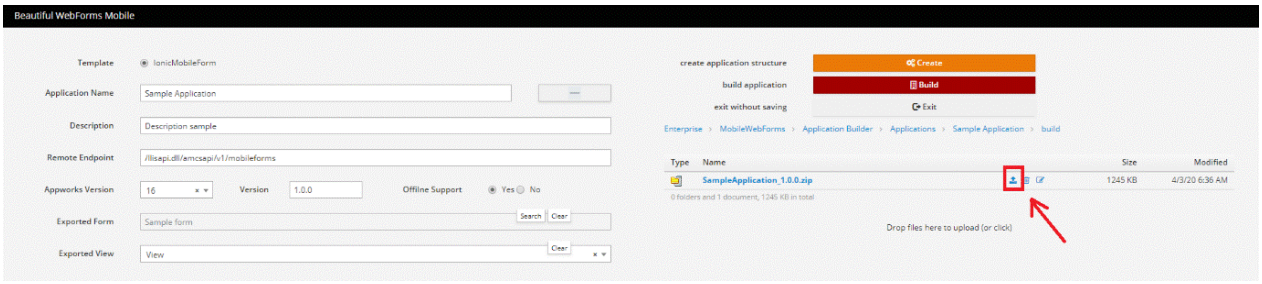
Once the application's structure has been created it will be possible to create an OpenText AppWorks Gateway deployable package by clicking on the "Build" button.



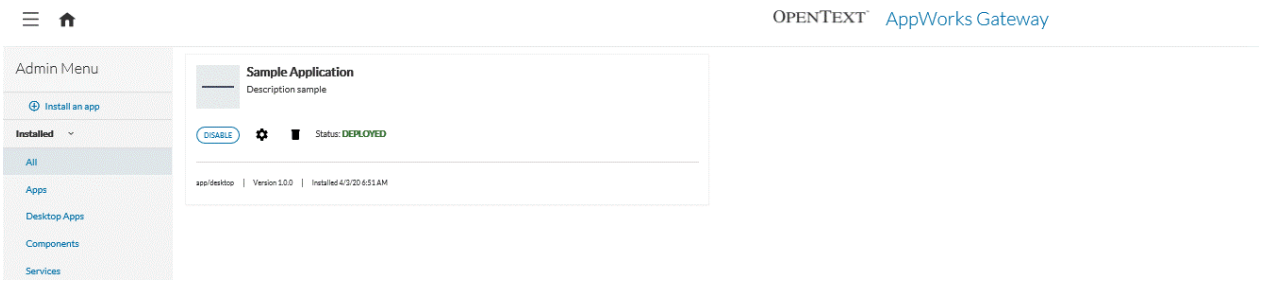
To upload the application to the OpenText AppWorks Gateway, enter the path and the authentication credentials of the destination OpenText AppWorks Gateway and click "connect".



Once connected to the OpenText AppWorks Gateway, the system will enable the user to deploy the application. Clicking on the deploy icon will automatically upload, install and enable the application on the OpenTextAppWorks Gateway.



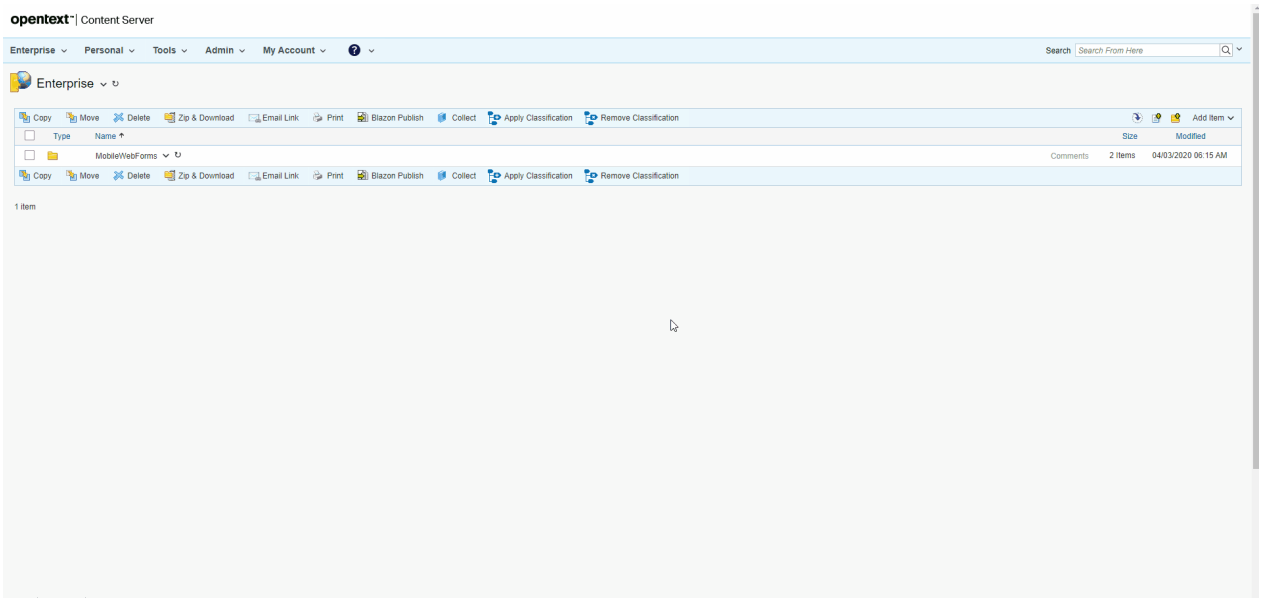
To verify the correctness of the process access the OpenText AppWorks Gateway and verify that in the "Installed" section the application to be distributed is present and enabled.



OpenText AppWorks Gateway

No information will be provided for installing and properly configuring the OpenText AppWorks Gateway. For installing ad configuring the OpenText AppWorks Gateway please refer to the official OpenText documentation.

The complete tour:



Extension: Remote WebForms

What is it?¶

Remote Beautiful WebForm is an extension package for Script Console *(/working/scriptconsole/base/)* that allows you to deploy a Beautiful WebForms powered webform created on Content Server on the Script Console engine.

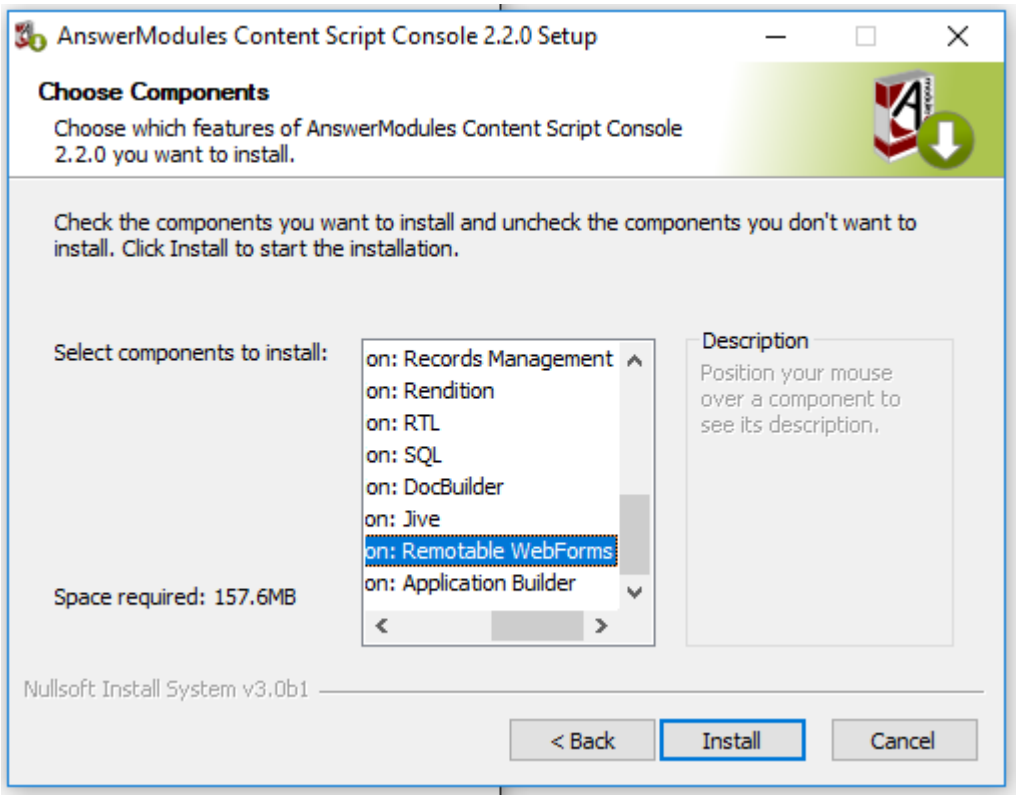
The main purpose of this extension is to simplify the process of gathering the contribution of users that do not have access to Content Server and synchronize these information back on Content Server. An other quite common scenario, is the off-line usage of Content Server webforms: the possibility of accessing, through a locally deployed Script Console instance, a copy of a Content Server webform, even when a connection with Content Server is not available.

In both the cases the information submitted through the remote webform are stored locally within the Script Console to be later synchronize back towards Content Server.

Extension setup¶

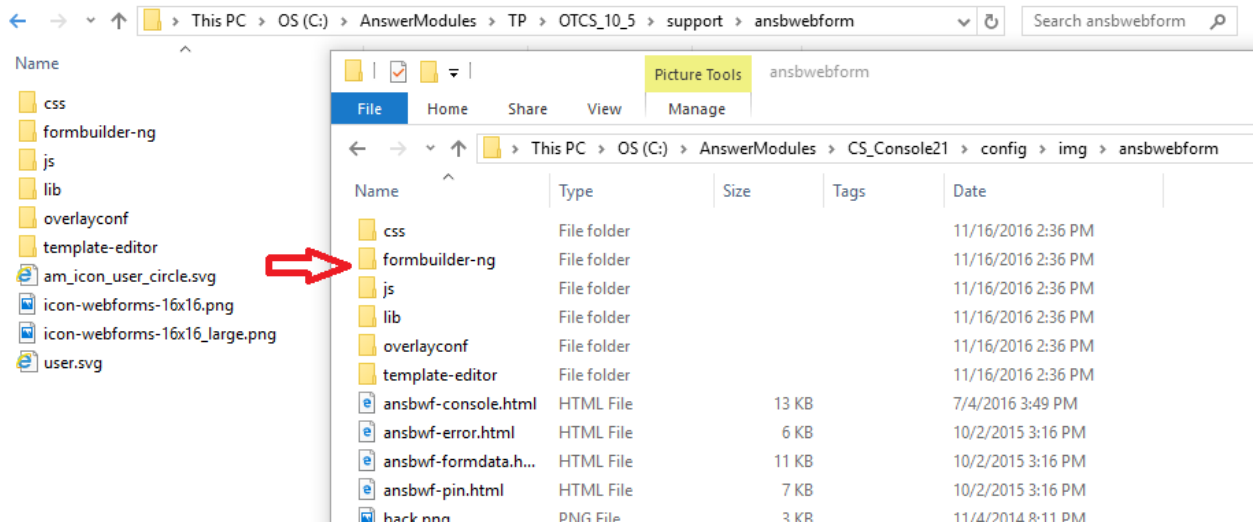
Installing the remote-webform extension package on a Script Console instance, is a straight forward procedure which consists of just two steps:

- Run the Script Console master installer and install the **Remotable WebForms** extension package



- Copy all the static resources from the Beautiful WebForms Module Support in:

```
<Script Console Home>\config\img\ansbwebform
```

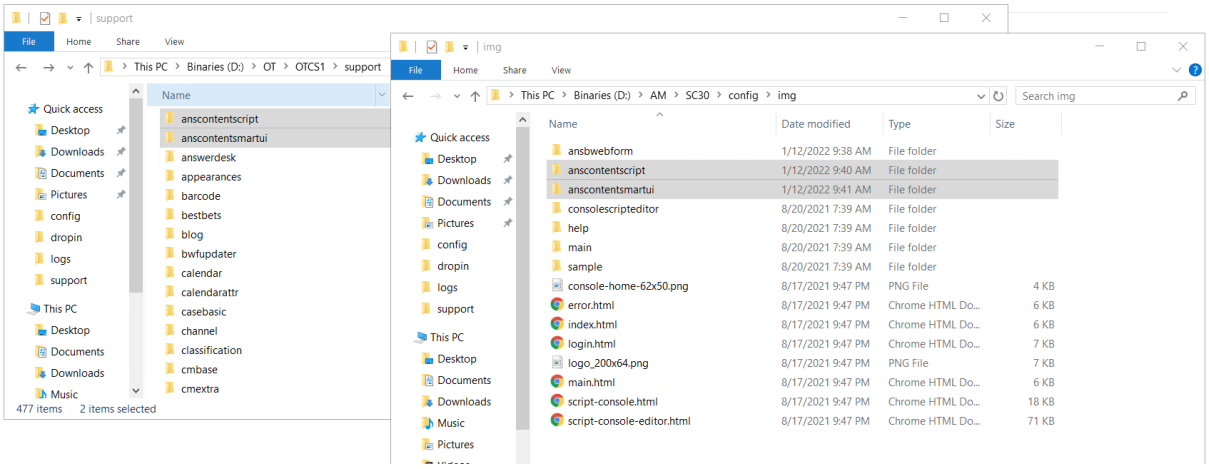


- Copy all the static resources from the Content Script Module Support (\support\anscontentscript) in:

```
<Script Console Home>\config\img\anscontentscript
```

- Copy all the static resources from the Module Suite for SmartUI Module Support (\support\anscontentsmartui) in:

```
<Script Console Home>\config\img\anscontentsmartui
```



Create remote package¶

Beautiful WebForms deployable packages can be created either programmatically, using the Content Script `forms` service or manually, through the Beautiful Webforms Studio application.

Using `forms.createExPackage` API¶

Content Script `forms.createExPackage` API can be used to programmatically create a deployable Beautiful WebForms remote package. The API can be used from within a Beautiful WebForms View CLEH script, or from any other Content Script object.

In most of the cases, if used within a stand-alone script, this API is used in conjunction with `forms.getFormInfo` or `forms.listFormData` APIs.

Properly initialize the form object

It's important that you keep in mind that when the `form` object is loaded using the `form` service it is not initialized. You can either initialize it as part of your script or rely on it's OnLoad CLEH for its proper initialization. Here below an example of how properly initialize the form object:

Minimum initialization required

```
def formNode= docman.getNodeByPath("Path:to:your:form")
form = formNode.getFormInfo()
forms.addResourceDependencies( form, true, true)
```

Initialization through the OnLoad script (if any)

```


def formNode= docman.getNodeByPath("Path:to:your:form")

form = formNode.getFormInfo()
def bwfView = docman.getNode(form.amViewId)
def onLoad = bwfView.childrenFast.find{it.name == "OnLoad"}

if(onLoad){
```

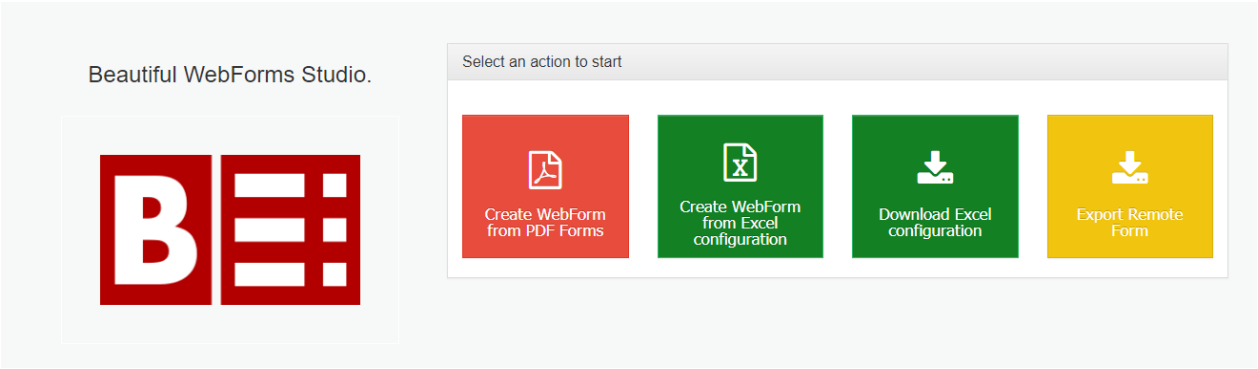
```
    docman.runContentScript(onLoad, binding)
}
```

```
forms.createExPackage(
                      Form form,   // The form to export
                      String name, // An alpha-numeric identifier for the package to be created
                      String instructions, // The instruction to be displayed to help the user filli
                      String nextUrl, // Where to redirect the user upon submission
                      Date validUpTo, // A date after which the form should no longer be available (
                      List<String> viewsToExport, // The names of the views you want to export as pa
                                                  // if null all the views will be exported
                      String pin,                 // An optional pin that can be used to protect the
                      CSDocument[] arrayOfDocuments // An optional list of documents to be exported
                     )
```

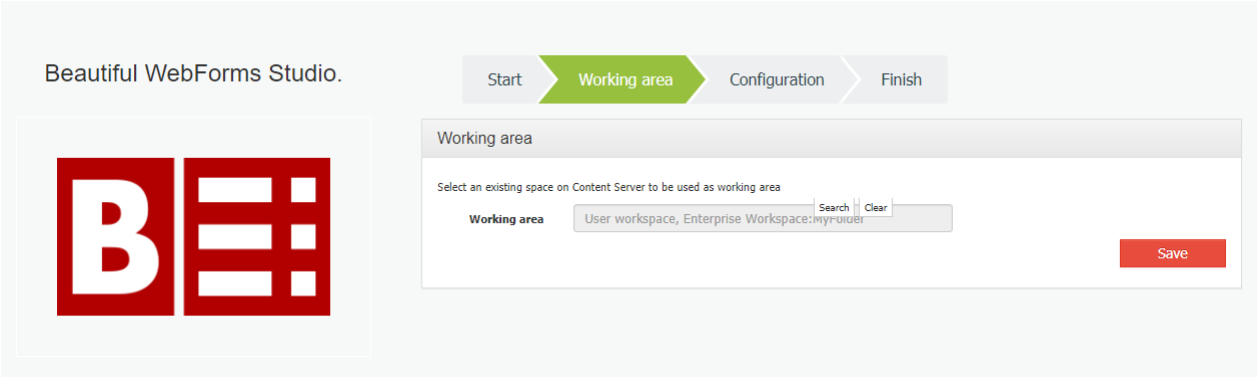
Using Beautiful Webforms Studio¶

Beautiful Webforms Studio which can be found at the following location: `Content Script Volume:CSTools:Beautiful WebForm Studio`

Among the possibilities offered the studio application can help you leveraging the `forms.createExPackage` through a simplified visual wizard. The first step is to select **Export Remote Form** among the available actions.

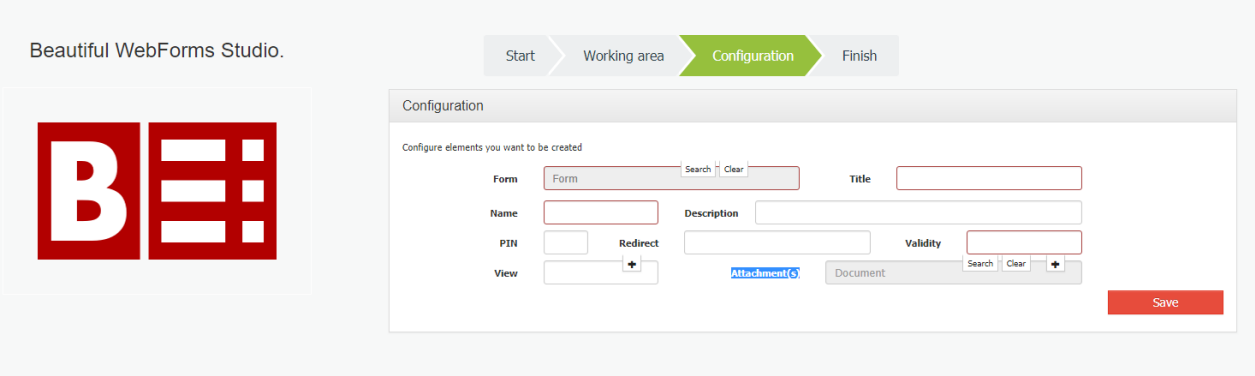


than you'll be asked for a space on Content Server to be used as the wizard workspace (where objects and content will be created):



finally you will be asked about export configuration parameters

- **Form**: the form object to be exported
- **Title**: the form's title as it will be displayed on the script console default dashboard
- **Name**: the export package name (should be an alpha-numeric value)
- **Description**: the form's description as it will be displayed on the script console default dashboard
- **PIN**: an optional PIN to be used in order to protect un-authorized access to the form on the console
- **Redirect**: an URL where to redirect user's navigation upon submission
- **View**: an optional list of views names to be exported
- **Attachment(s)**: an optional list of documents to be exported



upon submission the export package file will be created in the selected workspace.

How to deploy a Beautiful WebForms remote form package¶

The Beautiful WebForms remote form package is actually a .zip archive containing all objects necessary to the form (view files, scripts, templates, etc.).

You can manually extract its contents in a new folder inside:

```
<Script Console Home>\config\scripts\ext\forms\forms
```

for example:

```
<Script Console Home>\config\scripts\ext\forms\forms\myform
```

at this point, you should be able to access the form via the Script Console Dashboard, or via direct URL.

Synchronize form data back to Content Server¶

Form data submitted on Script Console can be synchronized back to Content Server in different ways which all are based on the same paradigm: the asynchronous exchange of information is based on data files.

Data files can be moved from the Script Console to Content Server no matter which transportation mechanism is used.

In the following paragraphs we will cover the most common scenarios.

Remote data pack files are produced on Script Console and sent over to Content Server¶

> **Script Console and Content Server can be isolated**
>
> In order to implement this scenario there is no need for the two systems to communicate each other.

In this scenario a local script is executed (or scheduled) on the Script Console in order to collect submitted data and prepare the exchange data files to be sent over Content Server.

The Remotable Beautiful WebForms extension for Script Console comes with several exemplar scripts of this kind that can be found at the following location:

```
<Script Console Home>\config\scripts\ext\forms
```

E.g **synchLocal.cs**

```
import groovy.json.JsonSlurper
import groovy.io.FileType
import java.util.zip.ZipOutputStream
import java.util.zip.ZipEntry
formsAvailable = []
system = context.getAttribute("system")
formRepository =  system.extensionRepositories.find{
    it.repoHome.name == 'forms'
}
formRepositoryDir = new File(formRepository.getAbsolutePath(), "forms")
formRepositoryDirLocal = new File(formRepository.getAbsolutePath(), "inout")
if(formRepositoryDir && formRepositoryDir.isDirectory()){
    def deleteFile = []
    formRepositoryDirLocal.eachFileRecurse(FileType.FILES){
        if(it.name.endsWith(".amf")){
            File newForm = new File(formRepositoryDir, it.name-'.amf')
            if(!newForm.mkdir()){
                return
            }
            def zipFile = new java.util.zip.ZipFile(it)
            zipFile.entries().each {
```

```
                ins = zipFile.getInputStream(it)
                new File(newForm, it.name) << ins
                ins.close()
            }
            zipFile.close();
            deleteFile << it
        }
    }
    deleteFile.each {
        it.delete()
    }
}
if (params.upload == 'true' && params.selform){
    list =[]
    list.addAll( params.selform)
    toBeDeleted = []
    list.each{ form->
        formRepositoryDir = new File(formRepository.getAbsolutePath(), "data/$form")
        if(formRepositoryDir && formRepositoryDir.isDirectory()){
            formRepositoryDir.eachFileRecurse(FileType.FILES){
                if(it.name == "data.amf"){
                    File dataPack = it.getParentFile()
                    String zipFileName = "${dataPack.name}.rpf"
                    File zipFile = new File(new File(formRepository.getAbsolutePath(), "temp"), zipF
                    ZipOutputStream zipOS = new ZipOutputStream(new FileOutputStream(zipFile))
                    zapDir(dataPack.path, zipOS, dataPack.path)
                    zipOS.close()
                    zipFile.renameTo(new File(formRepositoryDirLocal, zipFile.name))
                    toBeDeleted << dataPack
                }
            }
        }
    }
    toBeDeleted.each{
         it.deleteDir()
    }
}
def static zapDir(String dir2zip, ZipOutputStream zos, String stripDir) {
        File zipDir = new File(dir2zip)
        def dirList = zipDir.list()
        byte[] readBuffer = new byte[2156]
        int bytesIn = 0
        dirList.each {
            File f = new File(zipDir, it)
            if(f.isDirectory())
                zapDir(f.path, zos, stripDir)
            else {
                FileInputStream fis = new FileInputStream(f)
                ZipEntry anEntry = new ZipEntry(f.path.substring(stripDir.length()+1))
                zos.putNextEntry(anEntry)
                while((bytesIn = fis.read(readBuffer)) != -1) {
                    zos.write(readBuffer, 0, bytesIn);
                }
                fis.close();
            }
        }
}
redirect params.nextUrl
```

If you want to schedule this kind of scripts to be automatically executed by the Script Console you have to configure the job in the `cs-console-schedulerConfiguration.xml` file, which is a standard Quartz scheduler configuration file. You should find a sample job in there.

Here below a configuration example:

```
<?xml version="1.0" encoding="UTF-8"?>
<job-scheduling-data
    xmlns="http://www.quartz-scheduler.org/xml/JobSchedulingData"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.quartz-scheduler.org/xml/JobSchedulingData http://www.quartz-sche
    version="1.8">
    <pre-processing-commands>
        <delete-jobs-in-group>*</delete-jobs-in-group>  <!-- clear all jobs in scheduler -->
        <delete-triggers-in-group>*</delete-triggers-in-group> <!-- clear all triggers in scheduler
    </pre-processing-commands>
    <processing-directives>
        <!-- if there are any jobs/trigger in scheduler of same name (as in this
            file), overwrite them -->
        <overwrite-existing-data>true</overwrite-existing-data>
        <!-- if there are any jobs/trigger in scheduler of same name (as in this
            file), and over-write is false, ignore them rather then generating an error -->
        <ignore-duplicates>false</ignore-duplicates>
    </processing-directives>
    <schedule>
        <job>
            <name>PollJobSynchronization</name>
            <group>Synchronization</group>
            <job-class>com.answer.modules.cscript.console.scheduler.CommandLauncherJob</job-class>
            <job-data-map>
                <entry>
                    <key>script</key>
                    <value>ext/forms/synchLocal.cs</value>
                </entry>
                <entry>
                    <key>system</key>
                    <value>LOCAL</value>
                </entry>
            </job-data-map>
        </job>
        <trigger>
            <cron>
                <name>LaunchEvery1Minutes</name>
                <group>SynchronizationTriggerGroup</group>
                <job-name>PollJobSynchronization</job-name>
                <job-group>Synchronization</job-group>
                <start-time>2010-02-09T12:26:00.0</start-time>
                <end-time>2020-02-09T12:26:00.0</end-time>
                <misfire-instruction>MISFIRE_INSTRUCTION_SMART_POLICY</misfire-instruction>
                <cron-expression>0 * * ? * *</cron-expression>
                <time-zone>America/Los_Angeles</time-zone>
            </cron>
        </trigger>
    </schedule>
</job-scheduling-data>
```

Later on Content Server the data files are unpacked using the `forms` service from within a Content Script that can be either manually executed or scheduled.

E.g.

```
// remPack is a data pack file, how this file was obtained is not relevant.
// It may have been fetched from an email folder, a ftp server, a shared folder a cloud service,
// or even uploaded on Content Server using web-services, etc...
def packList = forms.getExPackageContent( remPack) // returns a Map<String, CSResource>

if(packList."data.amf"){
    def res = packList.find{it.key == "data.amf"}.value
    def form  = forms.deserializeForm(res.content.getText("UTF-8"))
```

```
    // The form object can be used for various purposes
    // Submitting the data back to Content Server
    forms.submitForm(form)

    // Starting a workflow
    def damageInvestigation = docman.getNodeByPath("Fleet Management:Workflows:Damage Ingestigation

    def inst = forms.startWorkFlow(damageInvestigation, form, "Form", "Damage Ingestigation - Veichl

    // Seding on a running workflow
    def task = workflow.getWorkFlowTask(form.getAmWorkID(), form.getAmSubWorkID(), form.getAmTaskID(

    forms.updateWorkFlowForm(
                            task, //The task
                            "Form Name", //The form name
                            form,  //The form object
                            true   // True if the task should be sent on
                           )

}
```

Form data are submitted directly from Script Console¶

Script Console and Content Server can't be isolated

In order to implement this scenario the two systems shall be able to communicate each other.

This scenario can be implemented executing or scheduling a script similar to the one reported here below on the Script Console:

```
import groovy.io.FileType

log.debug("Running Your Form Synch Job")

formsAvailable = []
system = context.get("system")
formRepository =  system.extensionRepositories.find{
    it.repoHome.name == 'forms'
}

//Synch up
formRepositoryDirParent = new File(formRepository.getAbsolutePath(), "data")
def toBeDeleted = []
formRepositoryDirParent.eachFileRecurse(FileType.DIRECTORIES){ formRepositoryDir->

    if(("yourform").equalsIgnoreCase(formRepositoryDir.name)){
        if(formRepositoryDir && formRepositoryDir.isDirectory()){
            formRepositoryDir.eachFileRecurse(FileType.FILES){
                if(it.name == "data.amf"){
                    formObj = forms.deserializeForm(it.text)
                    File dataPack = it.getParentFile()
                    try{
                        forms.submitForm(formObj)
                        toBeDeleted << dataPack
                    }catch(e){
                        log.error("Unable to synch data back to OTCS",e)
                    }
                }
            }
        }
    }
}
```

```
toBeDeleted.each{
    it.deleteDir()
}
```

If you want to schedule this kind of scripts to be automatically executed by the Script Console you have to configure the job in the `cs-console-schedulerConfiguration.xml` file, which is a standard Quartz scheduler configuration file. You should find a sample job in there.

Here below a configuration example:

```
<?xml version="1.0" encoding="UTF-8"?>
<job-scheduling-data
    xmlns="http://www.quartz-scheduler.org/xml/JobSchedulingData"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.quartz-scheduler.org/xml/JobSchedulingData http://www.quartz-sche
    version="1.8">
    <pre-processing-commands>
        <delete-jobs-in-group>*</delete-jobs-in-group>  <!-- clear all jobs in scheduler -->
        <delete-triggers-in-group>*</delete-triggers-in-group> <!-- clear all triggers in scheduler
    </pre-processing-commands>
    <processing-directives>
        <!-- if there are any jobs/trigger in scheduler of same name (as in this
            file), overwrite them -->
        <overwrite-existing-data>true</overwrite-existing-data>
        <!-- if there are any jobs/trigger in scheduler of same name (as in this
            file), and over-write is false, ignore them rather then generating an error -->
        <ignore-duplicates>false</ignore-duplicates>
    </processing-directives>
    <schedule>
        <job>
            <name>PollJobSynchronization</name>
            <group>Synchronization</group>
            <job-class>com.answer.modules.cscript.console.scheduler.CommandLauncherJob</job-class>
            <job-data-map>
                <entry>
                    <key>script</key>
                    <value>ext/forms/submitMyFormLocal.cs</value>
                </entry>
                <entry>
                    <key>system</key>
                    <value>LOCAL</value>
                </entry>
            </job-data-map>
        </job>
        <trigger>
            <cron>
                <name>LaunchEvery1Minutes</name>
                <group>SynchronizationTriggerGroup</group>
                <job-name>PollJobSynchronization</job-name>
                <job-group>Synchronization</job-group>
                <start-time>2010-02-09T12:26:00.0</start-time>
                <end-time>2020-02-09T12:26:00.0</end-time>
                <misfire-instruction>MISFIRE_INSTRUCTION_SMART_POLICY</misfire-instruction>
                <cron-expression>0 * * ? * *</cron-expression>
                <time-zone>America/Los_Angeles</time-zone>
            </cron>
        </trigger>
    </schedule>
</job-scheduling-data>
```

Smart Pages

Getting Started with Smart Pages¶

This guide provides a quick introduction to **Module Suite Smart Pages** and helps you get started with creating custom Smart View perspectives and interfaces.

What is Smart Pages?¶

Smart Pages is a Module Suite component that introduces new features for users who need extra flexibility when creating customized Smart View perspectives. It enables you to use Smart View in place of the Classic UI for your Content Server applications.

For a comprehensive overview, see the Introduction to Smart Pages.

Key Components¶

The Smart Pages extension includes:

- **Smart UI Tile Library** - A new set of Smart View tiles available in the Perspective Builder Widget Library
- **Smart Page Objects** - Content Server objects for creating presentation components
- **Content Script Data Sources** - Scripts that provide dynamic data to tiles
- **Smart View Overrides** - Low-coding capabilities to customize Smart View menus, columns, and actions
- **Beautiful WebForms Integration** - Embed web forms directly in Smart View perspectives

Quick Start Guide¶

1. Understanding the Basics¶

Start by reading the Introduction to Smart Pages to understand:

- What Smart Pages can do
- Use cases and examples
- Architecture overview

2. Working with Tiles¶

Learn about the available tiles and how to configure them:

- Smart UI Tiles - Complete tile reference and configuration guide
- Tile Communication - How tiles communicate with each other

3. Creating Smart Pages¶

Learn how to create and manage Smart Page objects:

- Smart Pages Object - Creation, properties, and the Smart Pages Editor IDE
- Smart Pages Editor - Learn how to design a Smart Page

4. Customizing Smart View¶

Extend Smart View functionality:

- Smart View Overrides - Custom menus, columns, and actions

5. Integrating WebForms¶

Embed forms in Smart View:

- WebForms Integration - Embedding Beautiful WebForms in Smart Pages

6. Advanced Features¶

Explore advanced capabilities:

- Smart Pages Commands - Using Smart View commands for advanced interactions

Smart Pages Fundamentals¶

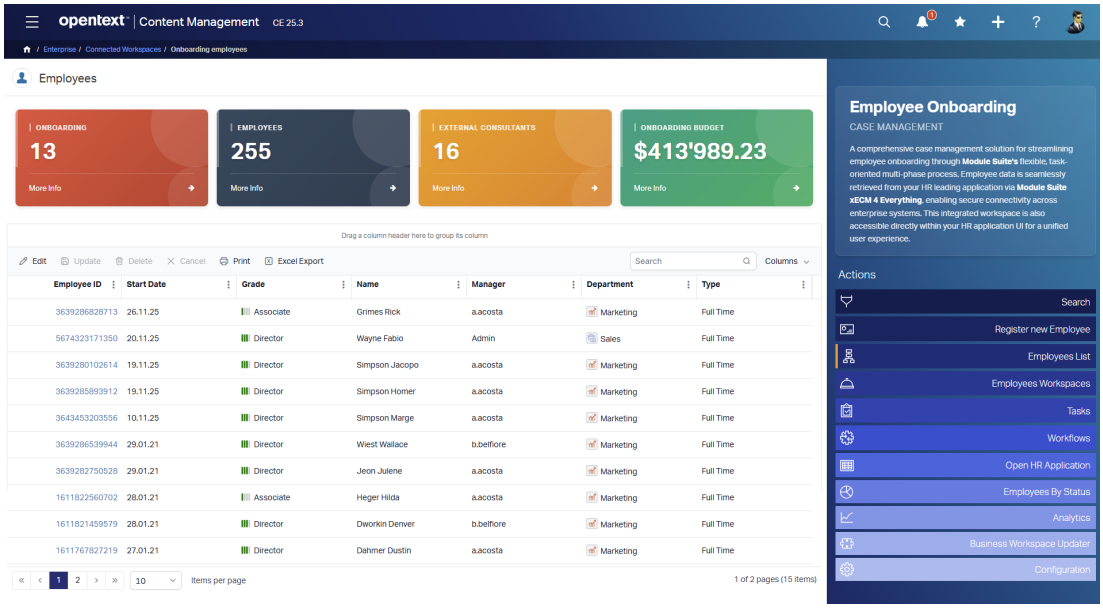
Introduction¶

OpenText Smart View (or **Smart View**) is an innovative UI Framework built around the concept of component reuse. However, the creation of new Smart View components ("**Tiles**") is an activity that can require a consistent amount of development efforts and requires in-depth knowledge of both the base libraries of the Smart View framework and the Smart View SDK.

For this reason, a limited number of ready-to-use components are available in Smart View.

What is "Smart Pages"?¶

The ultimate purpose of Smart Pages is to bring a significant improvement in the way Smart View perspectives are built, simplifying the task of organizing information in Smart View and enabling end-users to interact with much more productive interfaces.



An example of dashboard built with Smart Pages

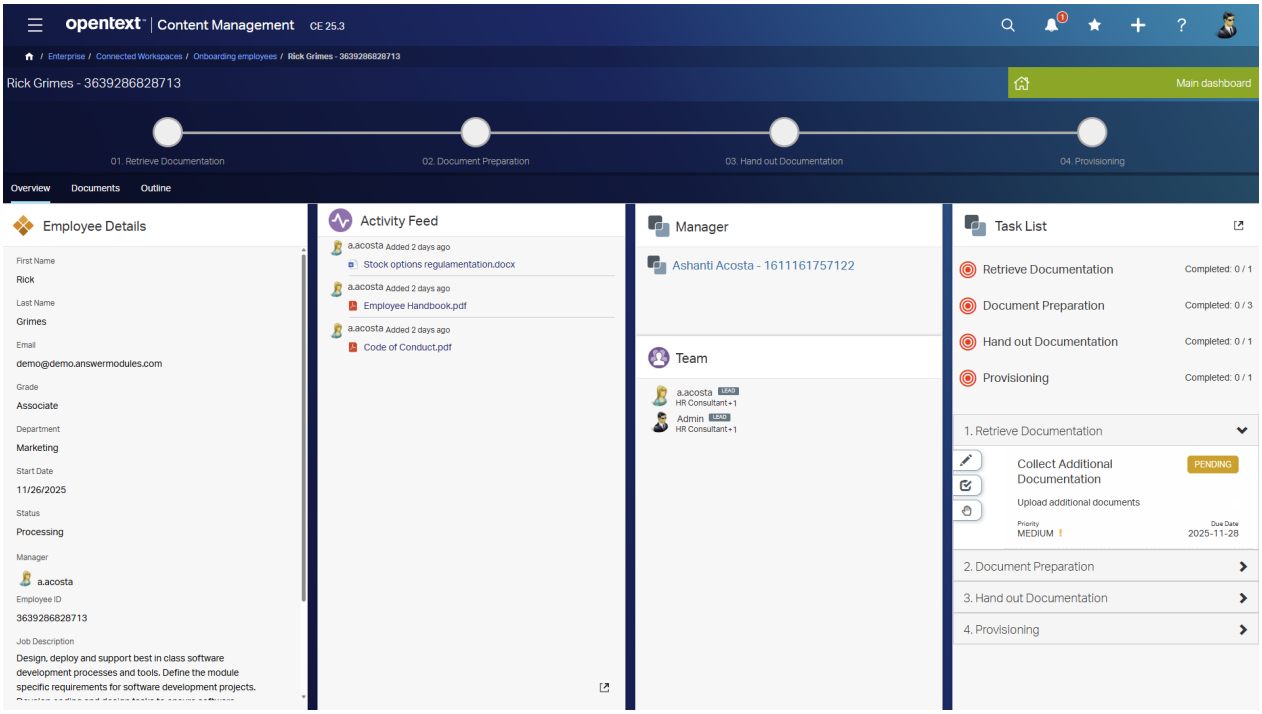
A few examples of the most relevant use-cases are:

- Building tailored web user interfaces compliant with Smart View look & feel and UX
- Building tailored Smart View dashboards for data analysis and reporting
- Publishing contextualized web forms in Smart View Perspectives
- Adding custom elements to standard elements of the Smart Views, such as menu entries and columns

Smart Pages: Usage Examples¶

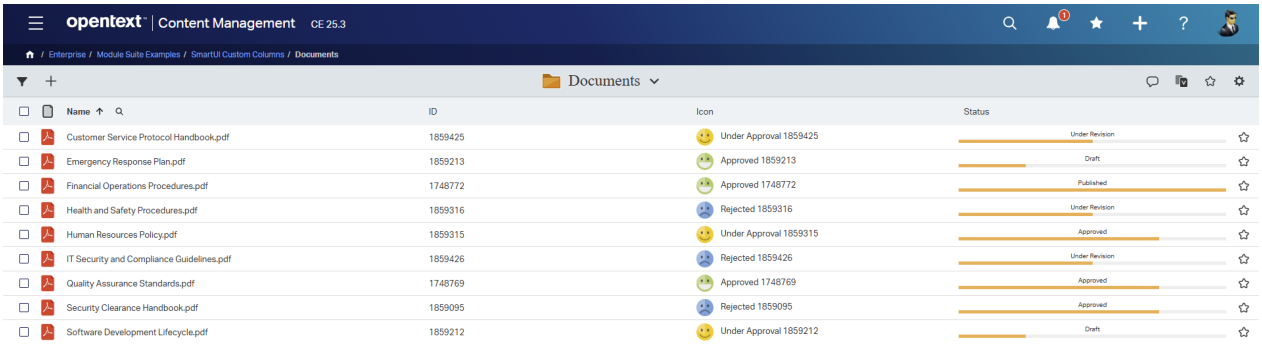
Tailored Perspectives with Custom Tiles¶

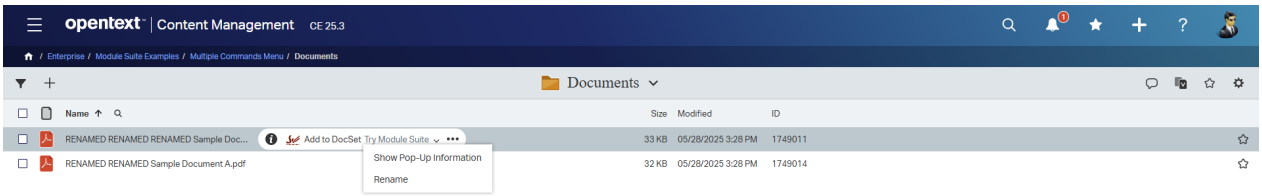
Smart Pages enables you to create custom perspectives that combine standard Smart View tiles with Module Suite tiles to create powerful, tailored interfaces.



Tailored Smart View Features (Menus, Columns)¶

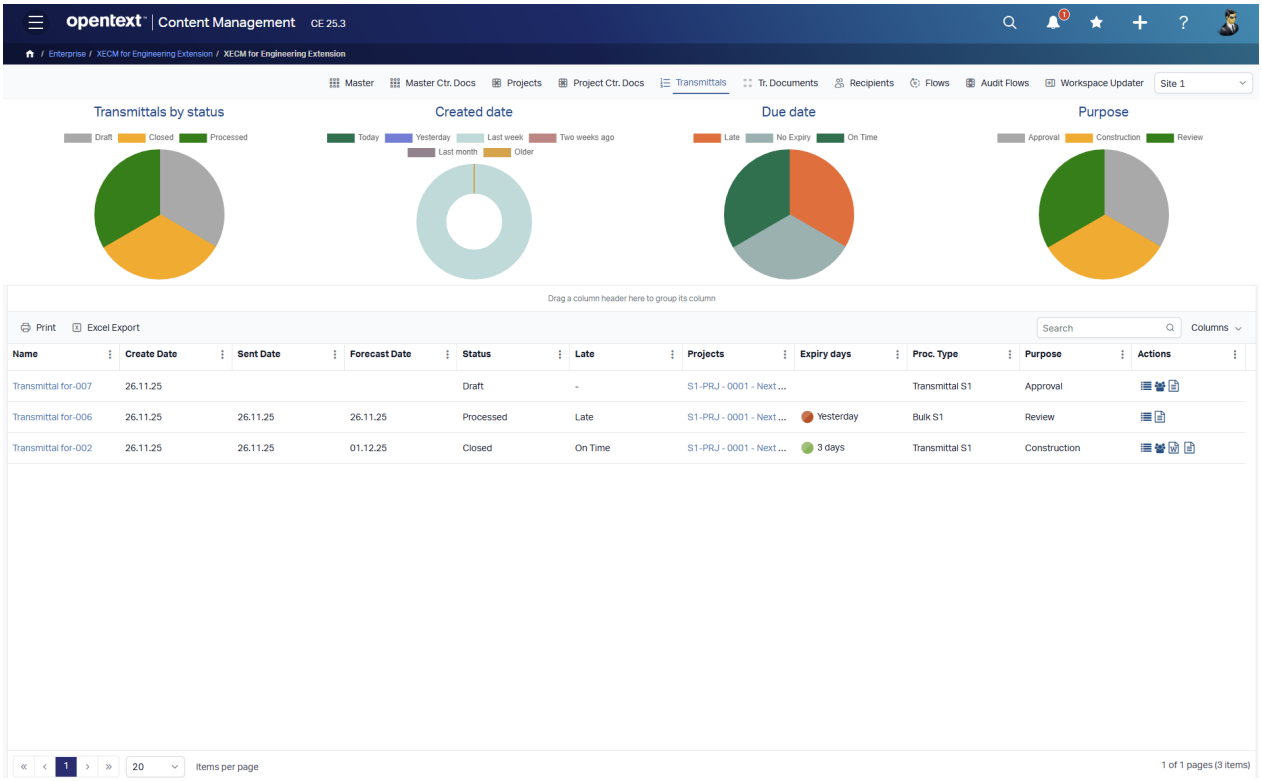
You can extend Smart View functionality by adding custom menu entries and columns to the standard Node Table tile, providing additional context and actions without requiring SDK development.





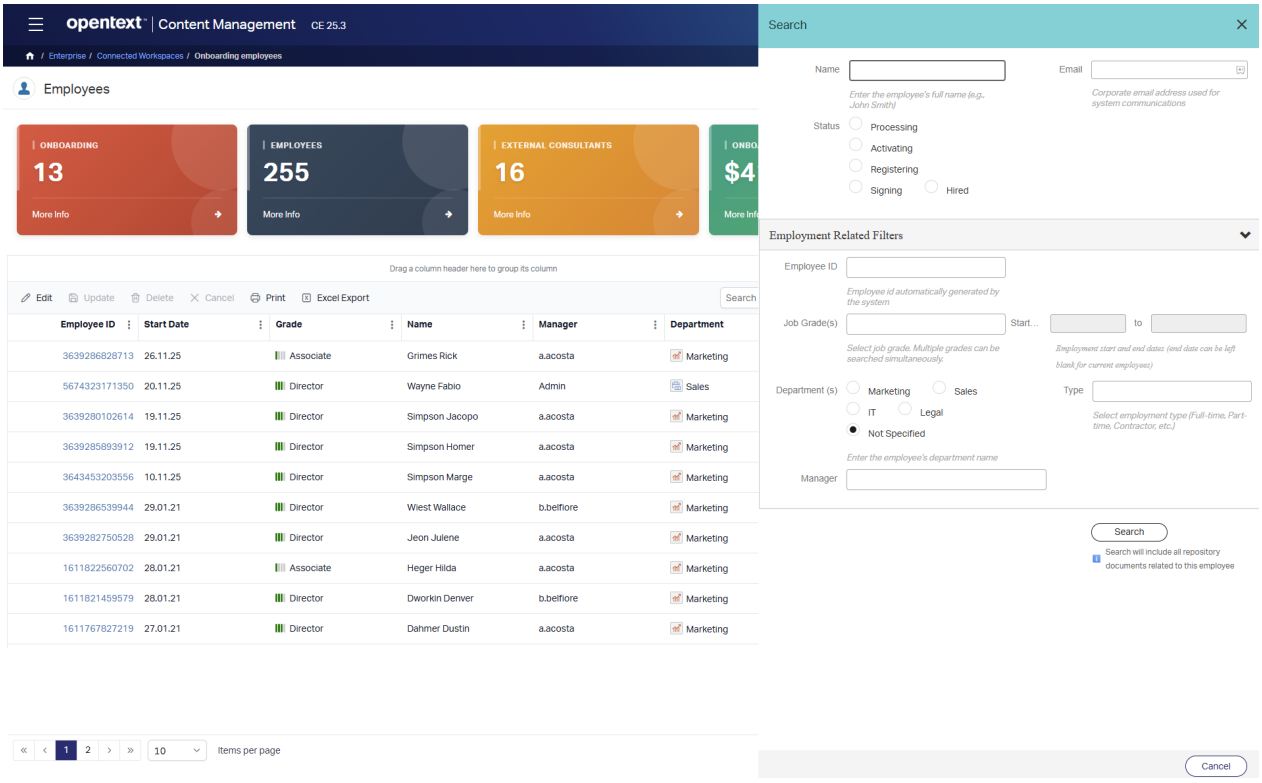
Standalone UIs¶

Smart Pages can be used to create standalone user interfaces that can be embedded in Smart View perspectives or accessed directly.



Embedded Forms¶

Smart Pages integrates seamlessly with Beautiful WebForms, allowing you to embed web forms directly into Smart View perspectives.



Smart Pages in the Module Suite Architecture¶

Smart Pages is a core component of the Module Suite architecture, providing a bridge between Content Script and Smart View.

```
flowchart TB
 subgraph MS["OT Content Management"]
        B["Content Script"]
        C["Smart View"]
        D["Smart Pages"]
  end
    B L_B_C_0@-- communicates --> C
    B --> D
    D -- embedded --> C

    style B fill:#fff,stroke:#D50000
    style C fill:#fff,stroke:#068fd7
    style D fill:transparent,stroke:#2962FF


    L_B_C_0@{ curve: natural }
```

Important

Smart Pages replace the Module Suite extension for Smart View which has been discontinued in Module Suite version 1.8.

What's in the Smart Pages Toolkit?¶

The Smart Pages toolkit includes:

- ✅ A custom **Smart UI Tile Library**, that can be used to extend the capabilities of the Smart UI perspective manager to build enhanced perspectives
- ✅ The actual **Smart Page object**, a new Content Server object that can be used to create presentation components
- ✅ **Smart View overrides**, that can be used to integrate additional features in the standard Nodes Table Tile (at the moment, menus and columns)
- ✅ **Smart Pages Beautiful WebForms integration**, used to bring the Module Suite web forms technology in the Smart View domain

Next Steps¶

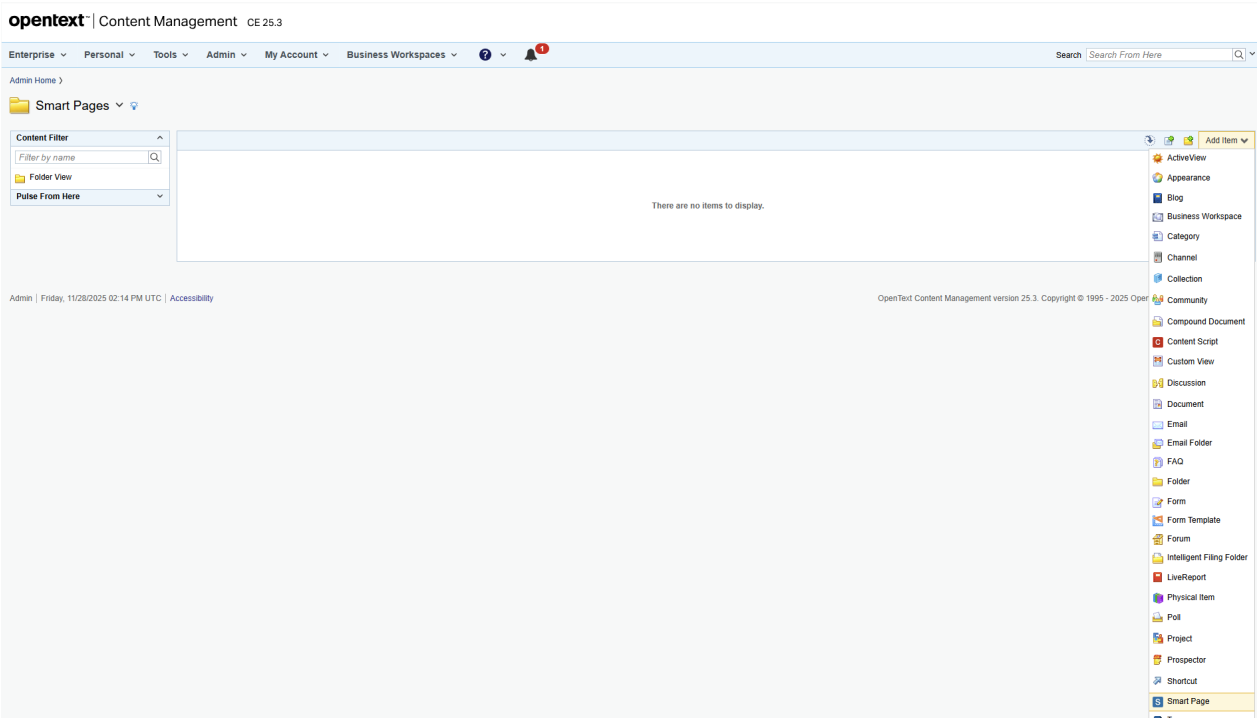
Now that you understand the fundamentals of Smart Pages, you can explore:

- Smart UI Tiles - Learn about the available tiles and how to configure them
- Smart Pages Object - Learn how to create and manage Smart Page objects
- Smart Pages Editor - Learn how to design a Smart Page
- Smart View Overrides - Learn how to customize Smart View menus and columns
- Tile Communication - Learn how tiles communicate with each other
- WebForms Integration - Learn how to embed Beautiful WebForms in Smart Pages

The Smart Pages Object¶

Overview¶

The **Smart Page** is a new Content Server object type that can be used to create presentation components. Smart Pages leverage the Content Script template engine's capabilities to create UI elements of any sort by adopting a rigorous MVC (Model-View-Controller) design pattern.



Creating a Smart Page¶

Prerequisites¶

Before creating a Smart Page, ensure that:

- Module Suite is installed and activated
- Smart Pages extension is enabled
- You have appropriate permissions to create Content Script and Smart Page objects

Creation Steps¶

1. **Navigate to the location** where you want to create the Smart Page
2. **Click "New"** and select **"Smart Page"** from the object types list
3. **Provide a name** for the Smart Page
4. **Click "Create"**

Understanding the Smart Page object¶

Smart Page: The MVC Pattern¶

Below is a conceptual diagram illustrating the Model-View-Controller (MVC) framework as applied to Smart Pages:

```
flowchart LR
    Controller["Content Script<br/>(Controller)"]
    Model["Data<br/>(Model)"]
    View["Smart Page<br/>(View)"]

    Controller -- "prepares data" --> Model
    Model -- "passed as input" --> View
    Controller -- "renders" --> View

    style Controller fill:#D50000,color:#ffffff,stroke:#B71C1C
    style Model fill:#ffffff,stroke:#ccc
    style View fill:#068fd7,color:#ffffff,stroke:#01579B
```

- **Smart Page (View)**: Implements the presentation, using the data provided.
- **Content Script (Controller)**: Prepares and supplies data (the model), and controls the rendering logic.
- **Data (Model)**: Passed from the Content Script to the Smart Page as input.

Smart Page pages are much more than simple html-views. They are **active** objects that can be used to create very complex applications. In order to implement all their additional

functionalities, Smart Pages pages are decorated with a set of information used by the Smart Page framework for determining how to render, and how to display the model-data within them.

In the image above a simplified representation of the information that constitutes a Smart Page page is highlighted:

- **(A)** View's versions: Smart Page pages are versioned document-class objects. Each version is, in the very end, nothing but a **Velocity** *(http://velocity.apache.org/)* template document (HTML code + template expressions).
- **(B)** For each version created with the PageBuilder's smart-editor the Smart Page framework archives the smart-editor view's "model" into an internal database (a Json file within the page). The smart-editor view's model is constituted by the list of the configurations used for each widget that build the page.
- **(C)** Page's properties: Smart Page pages are associated with a set of predefined properties persisted as the object's extended data. These properties are related just to the last page's version.
  The page's predefined properties are:

  1. PageBuilder mode used for creating the current page's version (either "source code" or "smart editor")
  2. The list of static "css" page's dependecies dynamically determined on the basis of the widgets used to build the page
  3. The list of static "javascript" page's dependecies dynamically determined on the basis of the widgets used to build the page
  4. The number of page's columns
  5. The identifier of the library of widgets used to build the page
  6. The ID of the page template (if any) associated to the view

Next Steps¶

- Smart Pages Editor - Learn how to design a Smart Page
- Learn about Smart View Overrides to customize Smart View behavior
- Explore Tile Communication for inter-tile communication
- See WebForms Integration for embedding forms in Smart Pages

The Smart Pages Object¶

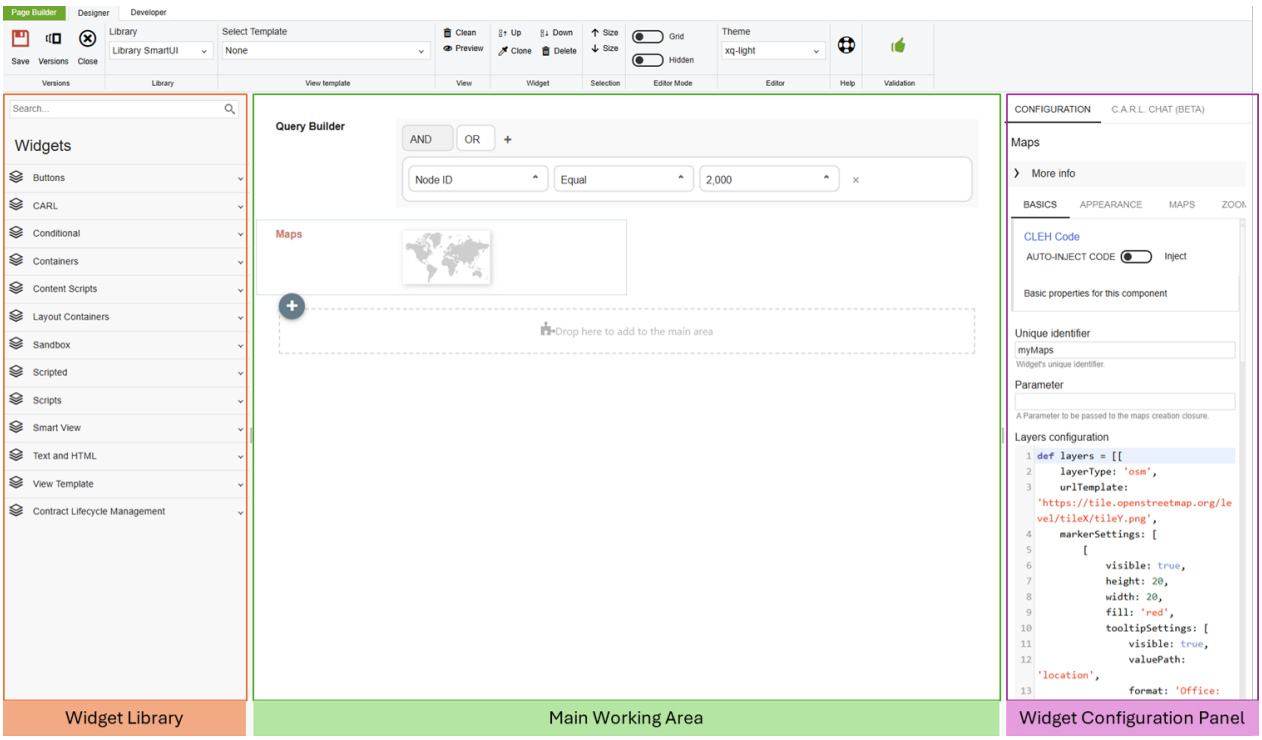
Editing a Smart Page: The Smart Pages Editor IDE¶

The Smart Pages Editor is the privileged IDE for Smart Pages. On the **first load** of an empty Smart Page, the Editor will initialize it with a default template structure. The page will then be available for further editing.

Layout¶

The IDE is composed of a set of areas and controls, with different purposes.

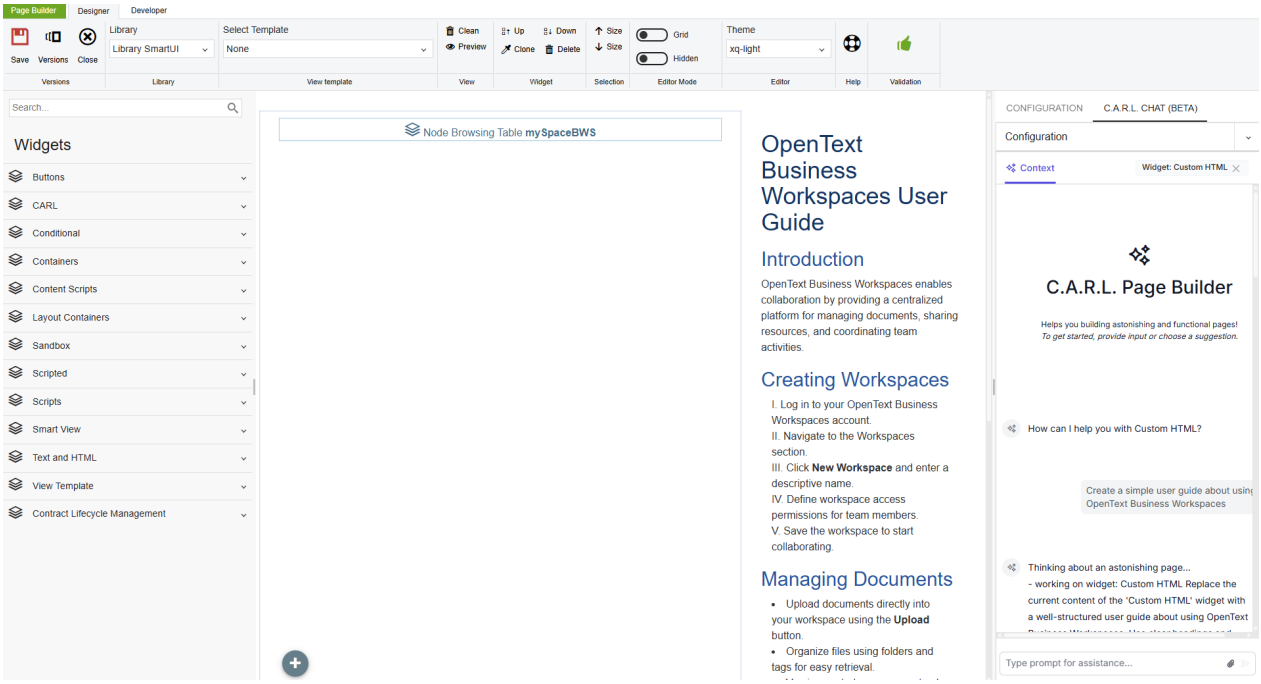
- The **Main Working Area** shows a preview of the current Smart Page, with the available content elements
- The **Widget Library** (on the left) features a set of predefined widgets and components, which can be easily dragged and dropped in the working area
- The **Configurator panel** (on the right) is linked to the element currently selected in the main working area



AI-Based Smart Page Builder¶

The AI-Based Smart Page Builder is an experimental feature that enables designers to create Smart Pages using natural language prompts. Instead of manually writing code or dragging and

dropping widgets, you can simply describe the page requirements, and C.A.R.L. (the AI assistant) will automatically generate the page for you.



How It Works¶

The AI-Based Smart Page Builder uses an **agentic workflow architecture** that processes your requests through a sophisticated multi-agent system:

1. **Coordinator Agent**: When you submit a request, a coordinator agent first analyzes it and creates an execution plan. This plan determines what needs to be done to satisfy your requirements, including:
2. Identifying which data sources need to be created or configured
3. Determining which widgets and components are most appropriate
4. Organizing the page layout and structure
5. **Component Agents**: Once the plan is determined, specialized component agents work in parallel to configure each element according to the coordinator's plan. Each component agent is dedicated to a specific component type and handles its configuration independently.

> **User Confirmation Required**
>
> Every change made by the AI requires your explicit confirmation before being applied to the Smart Page. This ensures you maintain full control over the page design and can review all AI-generated modifications before they are implemented.

Key Features¶

- **Natural Language Page Creation**: Simply describe what you want in the page, and the AI will create it automatically
- **UI Guidelines Support**: You can provide UI guidelines to guide the AI's design decisions, ensuring the generated pages match your design standards
- **Agent Configuration**: Fine-tune the AI's behavior through a dedicated configuration panel that allows you to adjust various agent parameters

Single Component Configuration¶

In addition to creating entire pages, you can also use the AI feature to configure individual components. When you select a specific component and interact with C.A.R.L., the AI will focus solely on that component's configuration without modifying the rest of the page. This allows you to:

- Refine individual component settings using natural language
- Get AI assistance for specific component properties
- Make targeted improvements without affecting other page elements

Context Support¶

The AI-Based Smart Page Builder supports adding files and images to the context, enabling the AI to:

- Understand visual requirements from reference images
- Process documentation or specifications provided as files
- Generate pages that match design mockups or examples

This context-aware capability allows for more accurate page generation based on visual and textual references.

> **C.A.R.L. Required**
>
> The AI-Based Smart Page Builder is an experimental feature that requires C.A.R.L. integration to be enabled and properly configured on your system. Without C.A.R.L., this feature will not be available.

Next Steps¶

- Learn about Smart View Overrides to customize Smart View behavior
- Explore Tile Communication for inter-tile communication
- See WebForms Integration for embedding forms in Smart Pages

WebForms in Smart Pages¶

Overview¶

Smart Pages integrates seamlessly with **Beautiful WebForms**, allowing you to embed web forms directly into Smart View perspectives. This integration brings the full power of the Module Suite web forms technology into the Smart View domain.

WebForms in Smart Pages

> **Placeholder for Image**
>
> Add a screenshot showing a web form embedded in a Smart View perspective

Why Embed WebForms in Smart Pages?¶

The main purpose of embedding Beautiful WebForms views into Smart View's tiles is to leverage the BWF framework as a primary input mechanism for your EIM applications. Integrating BWF into Smart View enables you to:

- **Collect and validate user input** within Smart View interfaces
- **Perform complex actions** through form submissions
- **Surface relevant business information** in highly interactive dashboards
- **Create seamless user experiences** without leaving Smart View

Prerequisites¶

Before embedding WebForms in Smart Pages, ensure:

- Module Suite is installed and activated
- Beautiful WebForms extension is installed
- Smart Pages extension is enabled
- You have appropriate permissions to create Form Templates and Views

Creating an Embeddable WebForm¶

Creating an *embeddable webform* is not different from creating any other webform in the system. The steps are:

1. **Create a Form Template object**
2. **Create a Beautiful WebForm View** associated to the Form Template created in the previous step

3. **Using the Beautiful WebForms Form Builder**, define your form (structure and layout)
4. **Create a standard Content Server Form object** and associate it to the previously created Form Template and Beautiful WebForm View

Embedding WebForms in Smart Pages¶

Method 1: Using Content Script Result Tile¶

The recommended approach is to use the **Include WebForms** widget. This widget requires a piece of Content Script business logic to be injected into the Controller script. The injection happens automatically whenever you save the widget's configuration.

The following example shows the Content Script snippet that initializes and renders the form:

```
model.data["view_12345"] = {
    // Retrieve the form node by ID
    formNode = docman.getNodeFast(12345)

    // Get form information and view
    form = formNode.getFormInfo()
    view = formNode.view

    // Set flag indicating the form is embedded in another application (e.g., using the XECM busines
    form.viewParams.am_in_page = model.data.serverOrigin != ''

    // Initialize field values
    form.aField.value = "Test Me"

    // Set view parameters that can be used by the form
    form.viewParams.aVariable = 1234
    form.viewParams.anOtherVariable = docman.getNodeByPath("My:Path").ID

    // Add resource dependencies (required if the form uses multiple views or subviews)
    forms.addResourceDependencies(form, true, true)

    // Return the form data structure with dependencies and rendered HTML
    return [
        id: formNode.ID,
        jsDeps: view.jsDeps + (form.viewParams.am_JsViewDependencies ?: []),
        cssDeps: view.cssDeps + (form.viewParams.am_CssViewDependencies ?: []),
        html: view.renderView(binding, form, true)
    ]
}
```

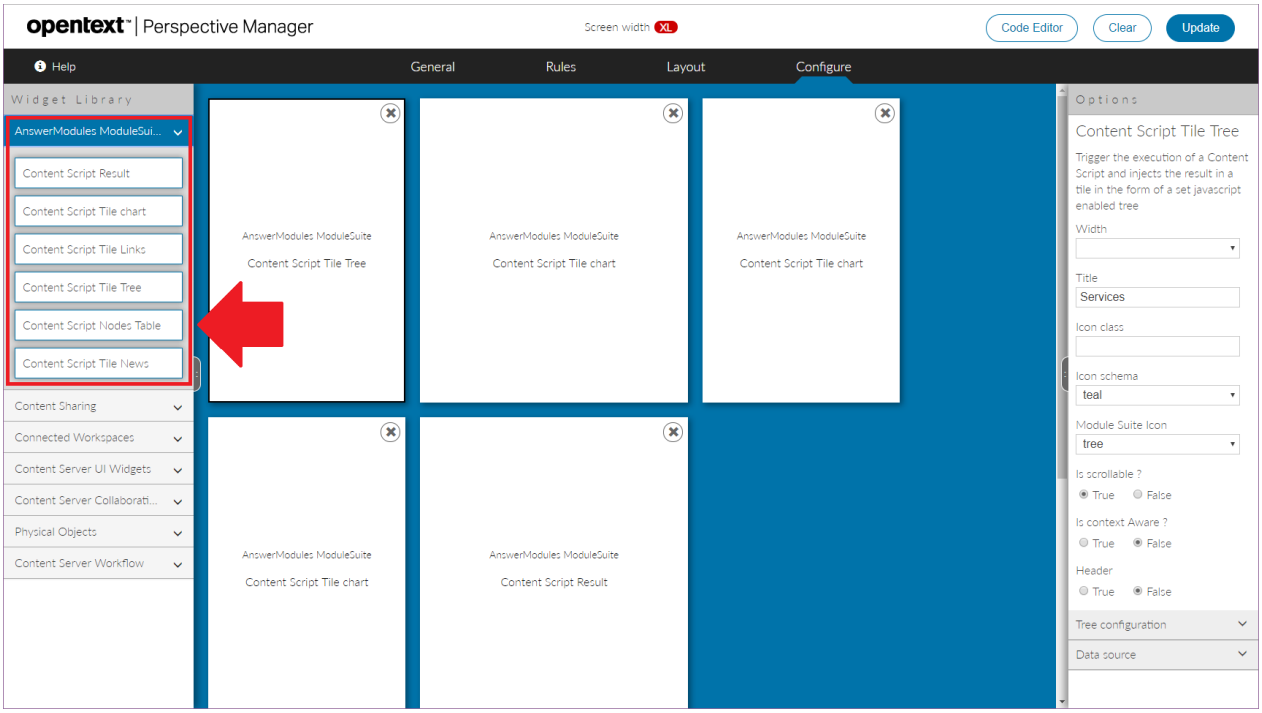
Next Steps¶

- Review Beautiful WebForms Documentation for detailed form building
- Explore Tile Communication for form-tile interactions
- Learn about Smart Pages Object for advanced page creation

Smart UI Tiles¶

Smart Pages enables an **extended set of pre-built tiles** to be available within the Smart View Perspective Manager tile library.

These can be added to perspectives just as any other out-of-the-box tile, and mixed & matched with other standard components.



Available Smart Page Tiles¶

The following tiles are available in the *AnswerModules Module Suite* section of the Widget Library:

- **Charts** - Used to create data visualization charts
- **Links** - Create a "menu" tile including a custom, dynamic list of links and actions
- **Tree** - Display a dynamic, lazy-loaded tree structure that can be used to display or access hierarchical data
- **Content Script Node Table** - Display an alternative version of the standard Node Browsing Table tile with customizable Content Script data source
- **News** - Display a list of expandable, actionable news items
- **Tiles** - An advanced version of the original Links Tile that allows for better usage of the screen space
- **Content Script Result** - A general-purpose tile that can be used to inject any output generated by a Content Script Data source or a Smart Page

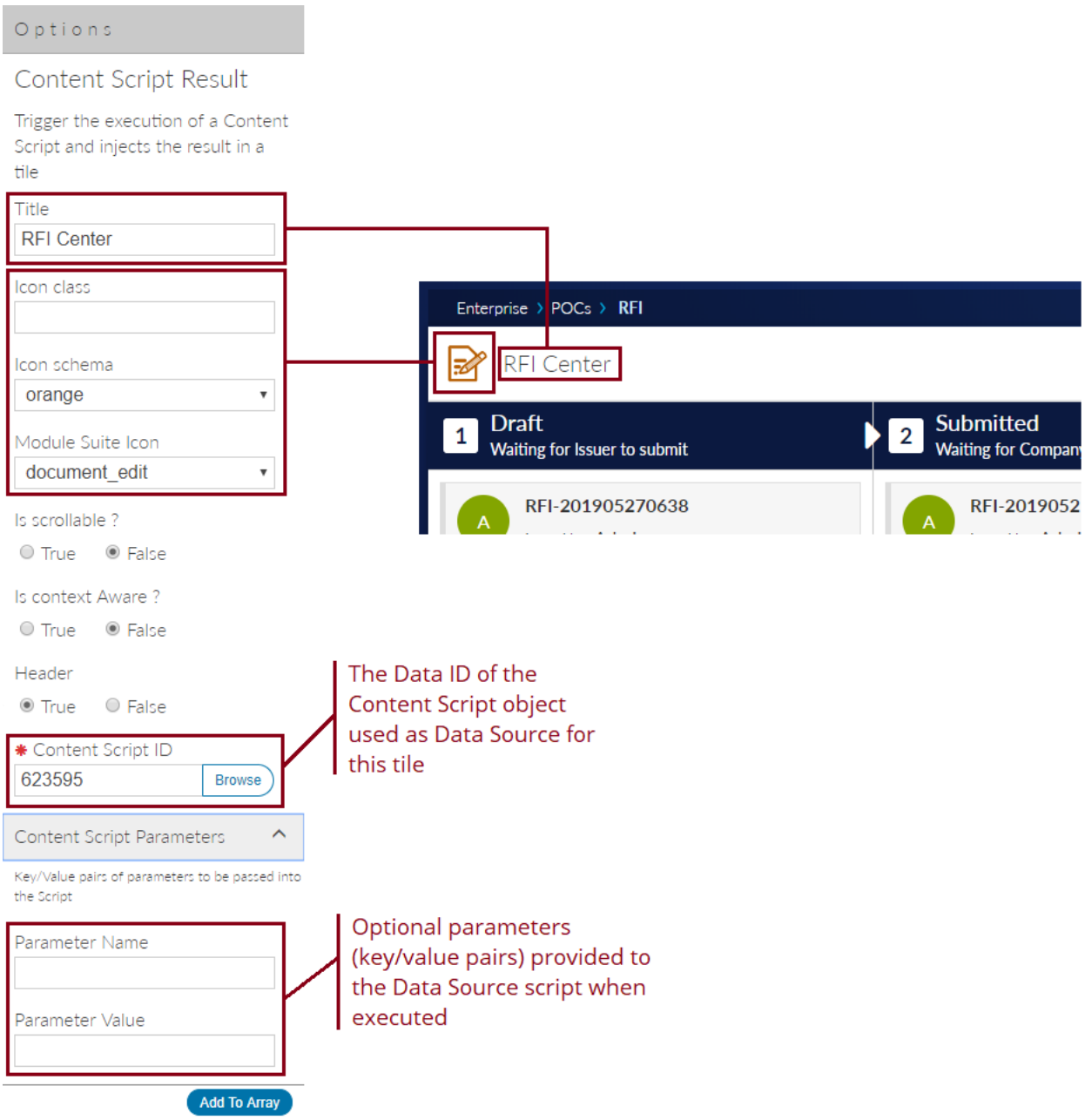
Tile Configuration¶

Module Suite tiles share some common configuration options, while other options are specific to single tiles.

Common Configuration Options¶

Common options include:

- **External frame configuration** (header, scrolling content, title, icon)
- **Data Source configuration** - All Module Suite tiles require a Content Script object that acts as a Data Source
- **Parameter passing** - Additional parameters can be passed to the script and will be available in the `params` variable



- **Icon Configuration** - When configuring the tile's icon, two different approaches are possible:

    1. **CSS Style Class**: Specify a CSS style class to apply to the icon element. This should define the rules needed to apply the desired icon.

    2. **Module Suite Icon Set**: Specify the name (and color scheme) of the desired icon among the ones available in the Module Suite icon set. See the Icon Reference Cheat Sheet for a full list of options.

- **Dynamic Configuration** - Module Suite tiles are designed to dynamically load their configuration from the same data source that supplies their data. This process is initiated by invoking the data source prior to the tile's rendering. To identify requests for

configuration-only data, the `widgetConfig` parameter is used. This parameter signals the data source that it's being called specifically for tile configuration.

Considerations for Dynamic Configuration

- **Flexibility vs. Initial Load Time**: While this feature offers increased flexibility, it does come with the trade-off of additional loading time for the initial data source call.
- **Optimized Data Source Responses**: It's advisable to configure the data source in a way that recognizes when it's being called solely for configuration purposes. Implementing strategies such as caching mechanisms or the use of static data can significantly expedite the configuration delivery.

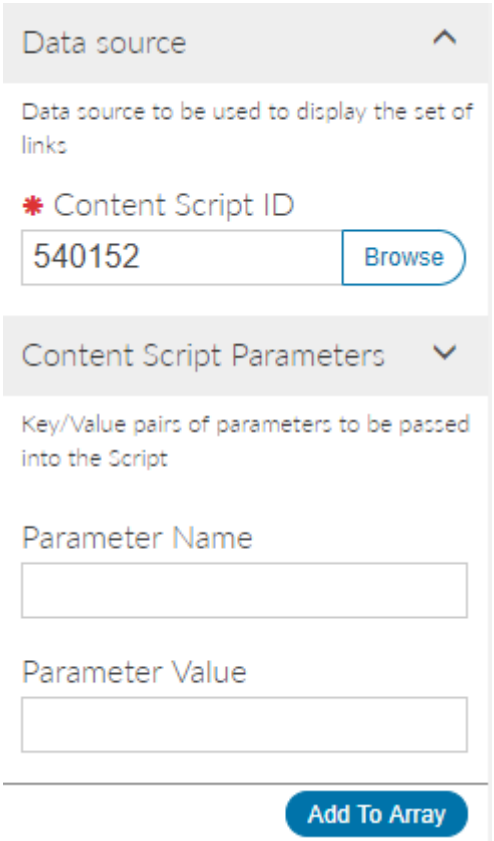
Content Script Data Sources¶

All Module Suite tiles require a Content Script object that will be executed when the tile content is created. This script acts as a **Data Source** for the tile, and allows to make its content dynamic.

The data source script receives:

- `params` - A map containing all parameters passed from the tile configuration
- `widgetConfig` - A boolean parameter indicating if the script is being called for configuration purposes

The script should return data in a format specific to the tile type being used.



Tile Library Reference¶

Content Script Tile Chart¶

The **Content Script Tile Chart** is a tile who's purpose is to create interactive charts within the Smart View. The data shown in the charts will be provided by a Content Script data source.

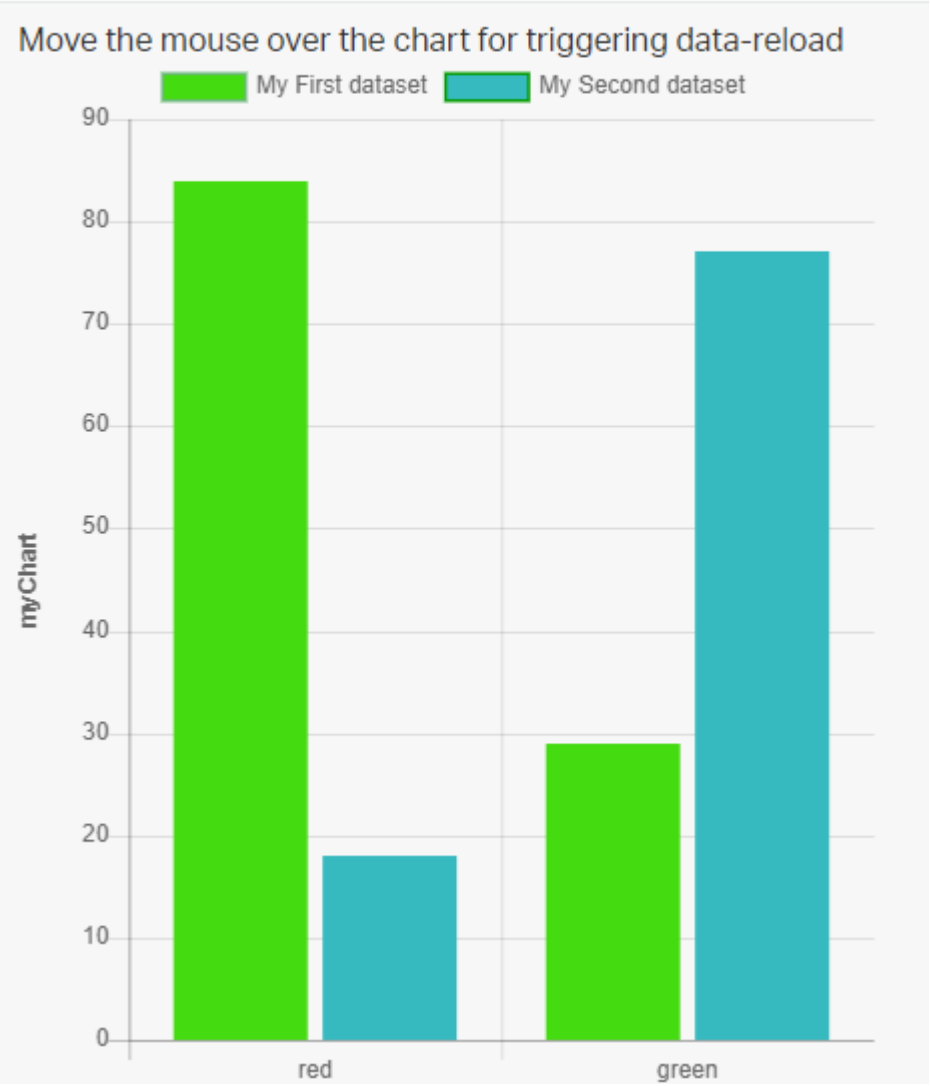
Chart tiles leverage two different javascript libraries:

- Chartist (supported for backward compatibility)
- Chart.js (suggested)

Depending on the selected chart type, the appropriate configuration has to be provided in JSON format.

ExampleDatasource

Module Suite Tile




```
def rand = new Random()

if(params.widgetConfig){

    json(widgetConfig:[
        reloadCommands:["updateChart"],
        html:"""


<small>Move the mouse over the chart for triggering data-reload</small>
<script>
    csui.onReady2([
        "csui/lib/jquery",
        "csui/lib/underscore",
        "csui/lib/radio"],
                function(jQ, _, Radio){

                    //Get the page message bus
                    var amChannel  = Radio.channel('ampagenotify');

                    //Get the chart
                    var chart = amChannel.request("ampages:myChart");
```

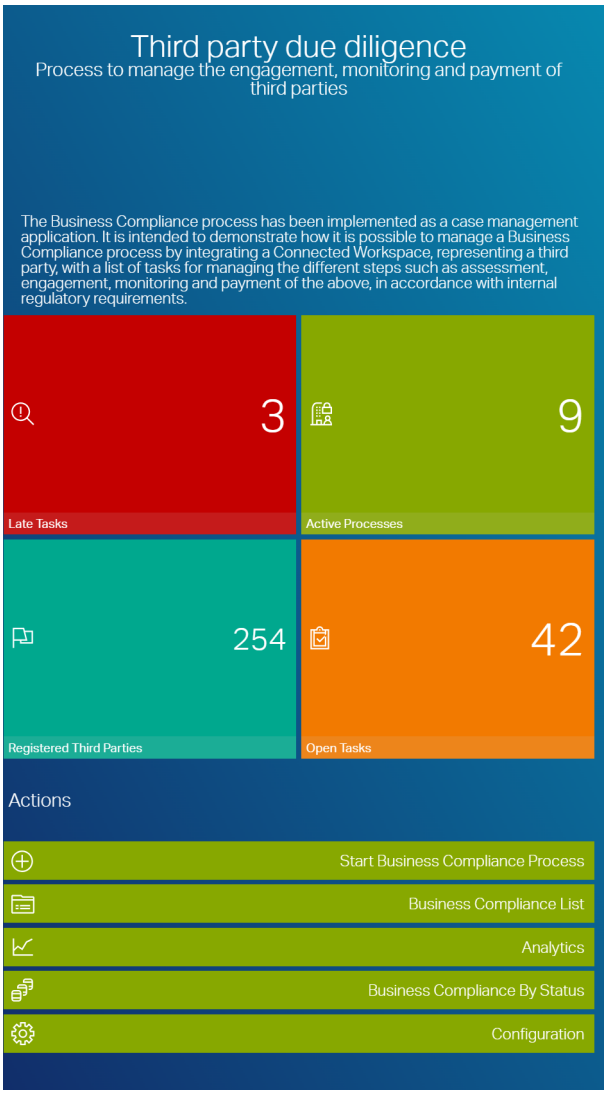
```
                    var canvas = jQ("#myChart");
                    canvas.unbind("click");
                    canvas.on("click", function (evt) {
                        var activePoints = chart.getElementsAtEvent(evt);
                        var vals = _.map(_.pluck(_.filter(chart.legend.legendItems, function(it){ re
                        if(!_.isUndefined(activePoints[0])){
                        var chartData = activePoints[0]['_chart'].config.data;
                        var idx = activePoints[0]['_index'];

                        var label = chartData.labels[idx];
                        var value = chartData.datasets[0].data[idx];
                        amChannel.trigger("updateChart", [ {name:"where_type", value:label} ]);

                    } else {
                        amChannel.trigger("updateChart", [ {name:"where_type", value:vals} ]);
                    }
                    });


                    canvas.hover(function(){
                        var self = jQ(this);
                        //jQ(".myChartLoader").removeClass("binf-hidden");
                        amChannel.trigger("updateChart", [{name:"filter", value:"first"}]);
                    });
                });
</script>"""
    ])
}else{

    json([

        type:"bar",
        data:
        [

            labels: ["red", "green"],
            datasets: [

                [
                    label: "My First dataset",
                    backgroundColor: "${AMBWFWidgetsLib.getBehaviour("ambwf","generateRandomHTMLColo
                    borderColor: "${AMBWFWidgetsLib.getBehaviour("ambwf","generateRandomHTMLColor",
                    data: [rand.nextInt(100), rand.nextInt(100)],

                ],
                [
                    label: "My Second dataset",
                    borderColor: "${AMBWFWidgetsLib.getBehaviour("ambwf","generateRandomHTMLColor",
                    backgroundColor: "${AMBWFWidgetsLib.getBehaviour("ambwf","generateRandomHTMLColo
                    data: [ rand.nextInt(100), rand.nextInt(100)],

                ]
            ]
        ],
        options: [
            maintainAspectRatio: false,
            title: [
                display: true,
                text: 'myChart',
                position: 'left'
            ],
            legend: [
                display: true,
                position: 'top'
            ],
            scales: [
                yAxes: [
                    [
                        ticks: [
```

```
                        beginAtZero:true
                    ]
                ]
            ]
        ]
    ])
}
```

Content Script Tile Tiles¶

The **Content Script Tile Tiles** is a tile meant to create a customizable list of clickable links and HTML Tiles. The data controlling the links is provided by the backing Content Script data source.

ExampleDatasource




```
app = runCS("am_businessCompliance")
if( params.widgetConfig ){
    json( widgetConfig : [
        reloadCommands : [ "updateTiles" ], // The widget will be refreshed when this command is
        html : """
```

```
<link type="text/css" rel="stylesheet" data-csui-required="true" href="${app.config.
<script>
csui.onReady2([
    'csui/lib/jquery',
    'csui/lib/underscore',
    'csui/lib/marionette',
    'csui/lib/radio',
    'csui/utils/commands',
    'csui/controls/side.panel/side.panel.view',
    'csui/controls/tile/behaviors/perfect.scrolling.behavior',
    'anscontentSmart View/utils/contexts/factories/scriptjsonresult.model.factory'
    ],
    function (jQ, _, Marionette, Radio, CommandsRegistry, SidePanelView, PerfectScro
        var ContentView = Marionette.View.extend({
            constructor: function ContentView(options) {
                this.widgetConfig = options.widgetConfig || {};
                this.options = options;
                Marionette.View.prototype.constructor.apply(this, arguments);
            },
            className: 'anscontentSmart View-tile-content-script',
            render: function () {
                var source;
                if (this.model) {
                    source = this.model.get('cssource');
                    if (!_.isUndefined(source)) {
                        var self = this;
                        csui.require(['csui/lib/jquery'], function (jQ) {
                            jQ(self.\$el).html(source);
                        });
                    }
                }
                return this;
            },
            className: 'amsui-exp-content-script',
            behaviors: {
                PerfectScrolling: {
                    behaviorClass: PerfectScrollingBehavior,
                    contentParent: ".am-Smart View",
                    suppressScrollX: true,
                    scrollYMarginOffset: 15,
                    scrollingDisabled: false
                }
            }
        });

        // Get the page message bus
        var amChannel = Radio.channel('ampagenotify');
        amChannel.off("tiles_action");
        amChannel.on("tiles_action", function (action, param) { //action = panel|123
            if( action.startsWith('panel') ){
                var scriptID   = undefined;
                var title      = "Action Panel";
                var panelWidth = 80;
                var params     = undefined;
                if( action.includes("|") ){
                    var tokens = action.split('|')
                    if( tokens.length >= 2){
                        if(jQ.isNumeric( tokens[1] )){ //panel|1234...
                            scriptID = tokens[1];
                            params = param;
                            if( tokens.length >= 3){
                                if(jQ.isNumeric( tokens[2] )){ //panel|1234|80
                                    panelWidth = tokens[2];
                                }else{ //panel|1234|My Title
                                    title = _.escape(tokens[2]);
                                    if( tokens.length >= 4){
                                        if(jQ.isNumeric( tokens[3] )){ //panel|1234|
                                            panelWidth = tokens[3];
                                        }
```

```
                                            }
                                        }
                                    }
                                }else{
                                    //panel|My title...
                                    title = _.escape(tokens[1]);
                                    if( tokens.length >= 3){
                                        if(jQ.isNumeric( tokens[2] )){ //panel|My Title|80
                                            panelWidth = tokens[2];
                                        }
                                    }
                                }
                            }
                            if( scriptID === undefined ){
                                scriptID = param;
                            }
                        }

                        if( jQ.isNumeric( scriptID ) ){
                            var context = amChannel.request("ampages:pageContext");
                            var scriptAttrs = { source: scriptID };
                            var script = context.getModel(ContentScriptModelFactory, { attri
                            if(params != undefined){
                                script.attributes.parameters = [{ name: "actionParams", valu
                            }
                            var slides = [
                                {
                                    title   : title,
                                    content : new ContentView({ model: script })
                                }
                            ];
                            script.fetch().then(function () {
                                var dialog = new SidePanelView(_.extend({
                                    sidePanelClassName : "amsui-Smart View-slide-panel-"+pan
                                    openFrom           : "right",
                                    slides             : slides
                                }));

                                dialog.show();
                                amChannel.on("tiles_panel:hide", function () {
                                        dialog.hide();
                                });
                            });
                        } else {
                            console.log("Error opening panel - invalid settings.")
                        }
                    };
                });
            });
        </script>
        <style>
        .binf-widgets [data-csui-widget_type=tilelinkstiles_content_script] .am-tile-content
            padding-right: 15px;
        }
        </style>
        """ ])
} else {

    json(
        data : [
            styleclass : "myStyleClass",
            rows : [
                [   // First row
                    styleclass : "",
                    size : 1, // The relative height of this row compared to other rows (default
                    tiles : [
                        [ // First Tile
                            size       : 1, // The relative size of this tile compared to others
                            styleclass : "",
```

```
                    html : """<div class="showcase-tiles-heading">
                            <div class="showcase-tiles-heading-main">Third party due dil
                            <div style="font-size: 0.6em;">Process to manage the engagem
                            </div>
                            """
                ]
            ]
        ],
        [   //Empty Row
            styleclass : "myStyleClass",
            size : 1, // The relative height of this row compared to other rows (default
            tiles : [
                [ // First Tile
                    size       : 1, // The relative size of this tile compared to others
                    styleclass : "myStyleClass",
                    html : """<div class="showcase-tiles-section"></div>"""
                ]
            ]
        ],
        [
            styleclass : "",
            size : 1, // The relative height of this row compared to other rows (default
            tiles : [
                [
                    size       : 1, // The relative size of this tile compared to others
                    styleclass : "",
                    html : """<div class="showcase-tiles-section" style="text-align: lef
                                The Business Compliance process has been implemented as
                                It is intended to demonstrate how it is possible to mana
                                integrating a Connected Workspace, representing a third
                                different steps such as assessment, engagement, monitori
                                internal regulatory requirements.
                            </div>"""
                ]
            ]
        ],
        [
            size : 3,
            tiles : [
                [   // First Tile
                    size : 1,
                    type : 'red', // Available types: red, green, blue, orange, teal, go
                    front : [
                        icon : "${img}anscontentSmart View/app/image/icons/windows10/whi
                        body : """3""",
                        body_text_align : 'right', // left, center, right (default)
                        body_text_size : 'jumbo', // small (90%), normal (100%), large (
                        title : "Late Tasks"
                    ],
                    back : [
                        icon : "${img}anscontentSmart View/app/image/icons/windows10/whi
                        title : "Late Tasks",
                        body_text_align : 'center',
                        body : """Potuit, iam districtum mucronem in proprium latus inpe
                                rector ausus miserabiles casus levare multorum. hinc ill
                                custodiam protectoribus mandaverat fidis."""
                    ]
                ],
                [   // Second Tile
                    size : 1,
                    type : 'green', // Available types: red, green, blue, orange, teal,
                    front : [
                        icon : "${img}anscontentSmart View/app/image/icons/windows10/whi
                        body : """9""",
                        body_text_align : 'right', // left, center, right (default)
                        body_text_size : 'jumbo', // small (90%), normal (100%), large (
                        title : "Active Processes"
                    ],
                    back : [
```

```
                        icon : "${img}anscontentSmart View/app/image/icons/windows10/whi
                        title : "Active Processes",
                        body_text_align : 'center',
                        body : """<table class="binf-table binf-table-condensed">
                                    <thead>
                                        <tr>
                                            <th>First Col</th>
                                            <th>Second Col</th>
                                        </tr>
                                    </thead>
                                    <tbody>
                                        <tr>
                                            <td>Some Data</td>
                                            <td>Other Data</td>
                                        </tr>
                                    </tbody>
                                </table>"""
                    ]
                ],
            ]
        ],
        [
            size : 3,
            tiles : [
                [
                    size : 1,
                    type : 'teal', // Available types: red, green, blue, orange, teal, g
                    front : [
                        icon : "${img}anscontentSmart View/app/image/icons/windows10/whi
                        body : """254""",
                        body_text_align : 'right', // left, center, right (default)
                        body_text_size : 'large', // small (90%), normal (100%), large (
                        title : "Registered Third Parties"
                    ],
                    back : [
                        icon : "${img}anscontentSmart View/app/image/icons/windows10/whi
                        title : "Registered Third Parties"
                    ]
                ],
                [
                    size : 1,
                    type : 'orange', // Available types: red, green, blue, orange, teal,
                    front : [
                        icon : "${img}anscontentSmart View/app/image/icons/windows10/whi
                        body : """42""",
                        body_text_align : 'right', // left, center, right (default)
                        body_text_size : 'jumbo', // small (90%), normal (100%), large (
                        title : "Open Tasks"
                    ],
                    back : [
                        icon : "${img}anscontentSmart View/app/image/icons/windows10/whi
                        title : "Open Tasks"
                    ]
                ],
            ]
        ],
        [
            styleclass : "myStyleClass",
            size : 1, // The relative height of this row compared to other rows (default
            tiles : [
                [ // First Tile
                    size       : 1, // The relative size of this tile compared to others
                    styleclass : "myStyleClass",
                    html : """<div class="showcase-tiles-section"> Actions </div>"""
                ]
            ]
        ],
        [
            size : 0,
```

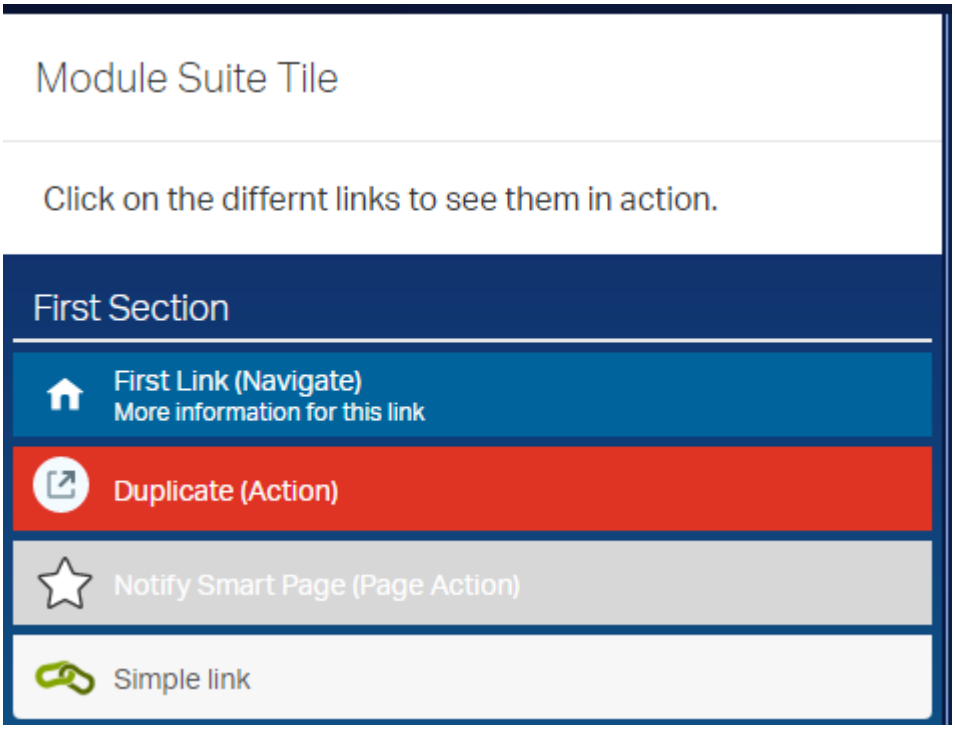
```
                    tiles : [
                        [
                            size : 12,
                            styleclass : "",
                            command : "tiles", // Custom command
                            action : "panel|Register New Third-Party|60",
                            params : app.config.pages.caseNew, //The action's parameter
                            newtab : false,
                            type : 'green',
                            front : [
                                icon : "${app.config.static.resourcesPath}add.svg",
                                body : "Start Business Compliance Process",
                            ]
                        ]
                    ]
                ],
                [
                    size : 0,
                    tiles : [
                        [
                            size : 12,
                            styleclass : "",
                            command : "cases", // Custom command
                            action : "z_changeMode",
                            params : "grid", //The action's parameter
                            newtab : false,
                            type : 'green',
                            front : [
                                icon : "${img}anscontentSmart View/app/image/icons/windows10/whi
                                body : "Business Compliance List",
                                //title : "Analytics"
                            ]
                        ]
                    ]
                ],
                [
                    size : 0,
                    tiles : [
                        [
                            size : 12,
                            styleclass : "",
                            command : "cases", // Custom command
                            action : "z_changeMode",
                            params : "stats", //The action's parameter
                            newtab : false,
                            type : 'green',
                            front : [
                                icon : "${img}anscontentSmart View/app/image/icons/windows10/whi
                                body : "Analytics",
                                //title : "Analytics"
                            ]
                        ]
                    ]
                ],
                [
                    size : 0,
                    tiles : [
                        [
                            size : 12,
                            styleclass : "",
                            command : "cases", // Custom command
                            action : "z_changeMode",
                            params : "kaban", //The action's parameter
                            newtab : false,
                            type : 'green',
                            front : [
                                icon : "${img}anscontentSmart View/app/image/icons/windows10/whi
                                body : "Business Compliance By Status",
                            ]
```

```
                        ]
                    ]
                ],
                [
                    size : 0,
                    tiles : [
                        [
                            size : 12,
                            styleclass : "",
                            command : "cases", // Custom command
                            action : "z_changeMode",
                            params : "conf", //The action's parameter
                            newtab : false,
                            type : 'green',
                            front : [
                                icon : "${app.config.static.resourcesPath}settings.svg",
                                body : "Configuration",
                            ]
                        ]
                    ]
                ],
                //Empty
                [
                    styleclass : "myStyleClass",
                    size : 1, // The relative height of this row compared to other rows (default
                    tiles : [
                        [
                            size       : 1, // The relative size of this tile compared to others
                            styleclass : "myStyleClass",
                            html : """<div class="showcase-tiles-section"></div>"""
                        ]
                    ]
                ],
            ]
        ]
    )
}
```

Content Script Tile Links¶

The **Content Script Tile Links** is a tile meant to create a customizable list of clickable links. The data controlling the links is provided by the backing Content Script data source.

ExampleDatasource




```
if(params.widgetConfig){

    json(widgetConfig:[
        reloadCommands:["updateLinks"],
        html:"""

<style>
div.ans-tile-content-linkstiles{
    background: linear-gradient(180deg, #122c69 0%, #078db3 100% );
    color:#fff;
    height:100%;
}

div.ans-tile-content-linkstiles > div.binf-list-group > a:nth-child(2),
div.ans-tile-content-linkstiles > div.binf-list-group > a:nth-child(6),
div.ans-tile-content-linkstiles > div.binf-list-group > a:nth-child(10){
    background:#00639b;
    color:#fff;
    border-radius:0px;
}
div.ans-tile-content-linkstiles > div.binf-list-group > a:nth-child(3),
div.ans-tile-content-linkstiles > div.binf-list-group > a:nth-child(7),
div.ans-tile-content-linkstiles > div.binf-list-group > a:nth-child(11){
    background:#df3324;
    color:#fff;
    border-radius:0px;
}
div.ans-tile-content-linkstiles > div.binf-list-group > a:nth-child(4),
div.ans-tile-content-linkstiles > div.binf-list-group > a:nth-child(8),
div.ans-tile-content-linkstiles > div.binf-list-group > a:nth-child(12){
    background:#008485;
    color:#fff;
    border-radius:0px;
}

</style>
<div style="padding:20px; background-color:white;margin-bottom:10px;color:#333" >
Click on the differnt links to see them in action.
</div>
<script>
```

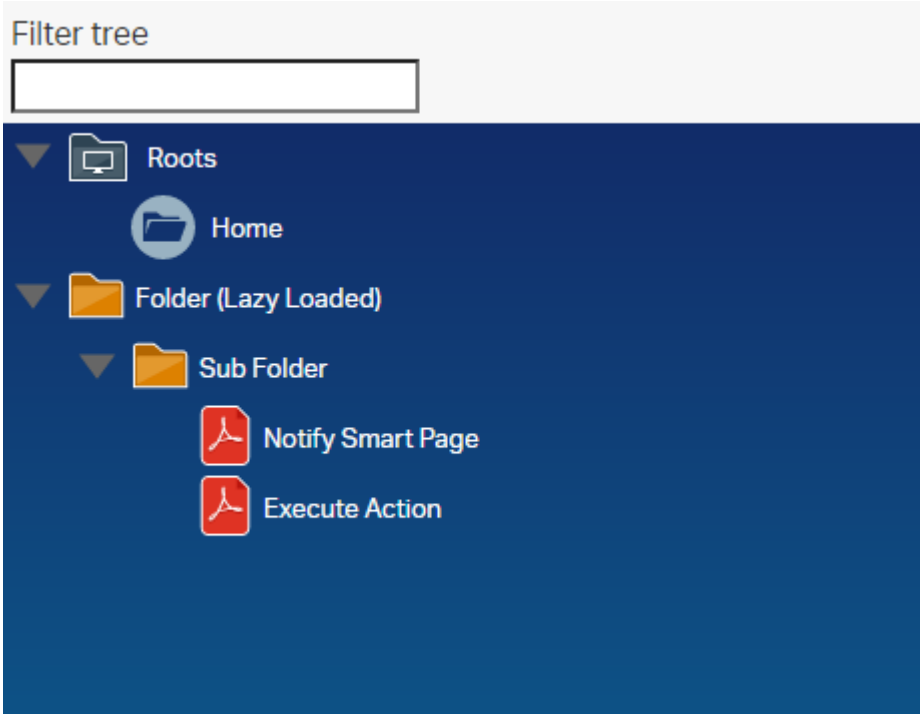
```
        csui.onReady2([    'csui/lib/underscore',
            'csui/lib/backbone',
            'csui/lib/jquery',
            'csui/lib/radio'],
            function(_,Backbone, jQ, Radio){
                var amChannel = Radio.channel("ampagenotify");
                amChannel.on("smartPage_action", function(action,param){
                    console.log("GOT Page Action request. Action: "+action+ " parameter: "+param);
                });

            });
    </script>
    """
        ])
    }else{

        retVal =
            [
                data:[
                    links:[
                        [
                            issection:true,
                            name:"First Section",
                        ],
                        [
                            issection:false,
                            icon:"csui-icon-home",
                            name:"First Link (Navigate)",
                            desc:"More information for this link",

                            url:"#", //If action != null url must be set equal to #
                            action:"navigate", //Will trigger a browse action of the current view
                            params:"2000", //The DataID of the node you wanto to navigate to

                        ],
                        [
                            issection:false,
                            icon:"icon-tileExpand icon-perspective-open",
                            name:"Duplicate (Action)",

                            url:"#", //If action != null url must be set equal to #
                            action:"notify", //Will trigger the execution of the command below
                            command:"updateLinks", //The action to execute
                            params:"duplicate", //The action's parameter, this value will be passed

                        ],
                        [
                            issection:false,
                            icon:"icon-socialFavOpen",
                            name:"Notify Smart Page (Page Action)",

                            url:"#", //If action != null url must be set equal to #
                            command:"smartPage", //The SmartPage(s) to notify
                            action:"updatePage",   //The action to execute
                            params:"2000" //The action's parameter

                        ],
                        [
                            issection:false,
                            am_icon:"am_icon_link",
                            am_icon_schema:"am_icon_green",
                            name:"Simple link",

                            url:"http://www.answermodules.com",
                            newtab:true

                        ]

                    ]
```

```
                ]
            ]

        if(params.tile == "duplicate"){
            retVal.data.links += retVal.data.links[-5].clone()
            retVal.data.links += retVal.data.links[-5]
            retVal.data.links += retVal.data.links[-5]
            retVal.data.links += retVal.data.links[-5]

            retVal.data.links[-4].name = "Second Section"
        }else if(params.tile == "triple"){
            retVal.data.links += retVal.data.links[-5].clone()
            retVal.data.links += retVal.data.links[-5]
            retVal.data.links += retVal.data.links[-5]
            retVal.data.links += retVal.data.links[-5]

            retVal.data.links[-4].name = "Second Section"

            retVal.data.links += retVal.data.links[-4].clone()
            retVal.data.links += retVal.data.links[-4]
            retVal.data.links += retVal.data.links[-4]
            retVal.data.links += retVal.data.links[-4]

            retVal.data.links[-4].name = "Third Section"
        }

        json(
            retVal
        )


    }
```

Content Script Tile Tree¶

The **Content Script Tile Tree** creates an interactive tree structure with nodes that can be expanded and collapsed. The tree structure uses a Content Script data source for the initial data and for subsequent ajax data load calls.

ExampleDatasource




```
if(params.widgetConfig){
    json( [ id           : 2,
        widgetConfig : [
            tileLayoutClasses  : "",
            tileContentClasses : "",
            reloadCommands     : ["updateTree"],
            root               : 2000,
            plugins            : [ "wholerow" ],
            theme              : [ 'name': 'proton',
                                   'responsive': true ],
            html               : """
<style>
div.ans-tile-tree{
background: linear-gradient(180deg, #122c69 0%, #078db3 100% );
color:#fff;
height:calc(100vh - 222px);
font-size:13px !important;
}
.binf-widgets .jstree-proton .jstree-icon.csui-icon-node-task {
background-image:url('${img}csui/themes/carbonfiber/image/icons/mime_task.svg')
}
.binf-widgets .jstree-proton .jstree-icon.mime_pdf{
background-image:url('${img}csui/themes/carbonfiber/image/icons/mime_pdf.svg')
}
.jstree-anchor small{
font-size:.9em;
font-style:italic;
}
</style>
<div class="am-form-text-input" style="margin-top: 1px;padding: 5px 0px;">
    <label class=" control-label col-form-label am-form-text-input-label   am-form-label-top" st
    <div class="am-form-input-wrap" style="padding: 0 5px;">
        <input id="filter" type="text" placeholder="" class="form-control" style="border-radius:
    </div>
</div>

<script>
    csui.onReady2([    'csui/lib/underscore',
        'csui/lib/backbone',
        'csui/lib/jquery',
        'csui/lib/radio'],
```

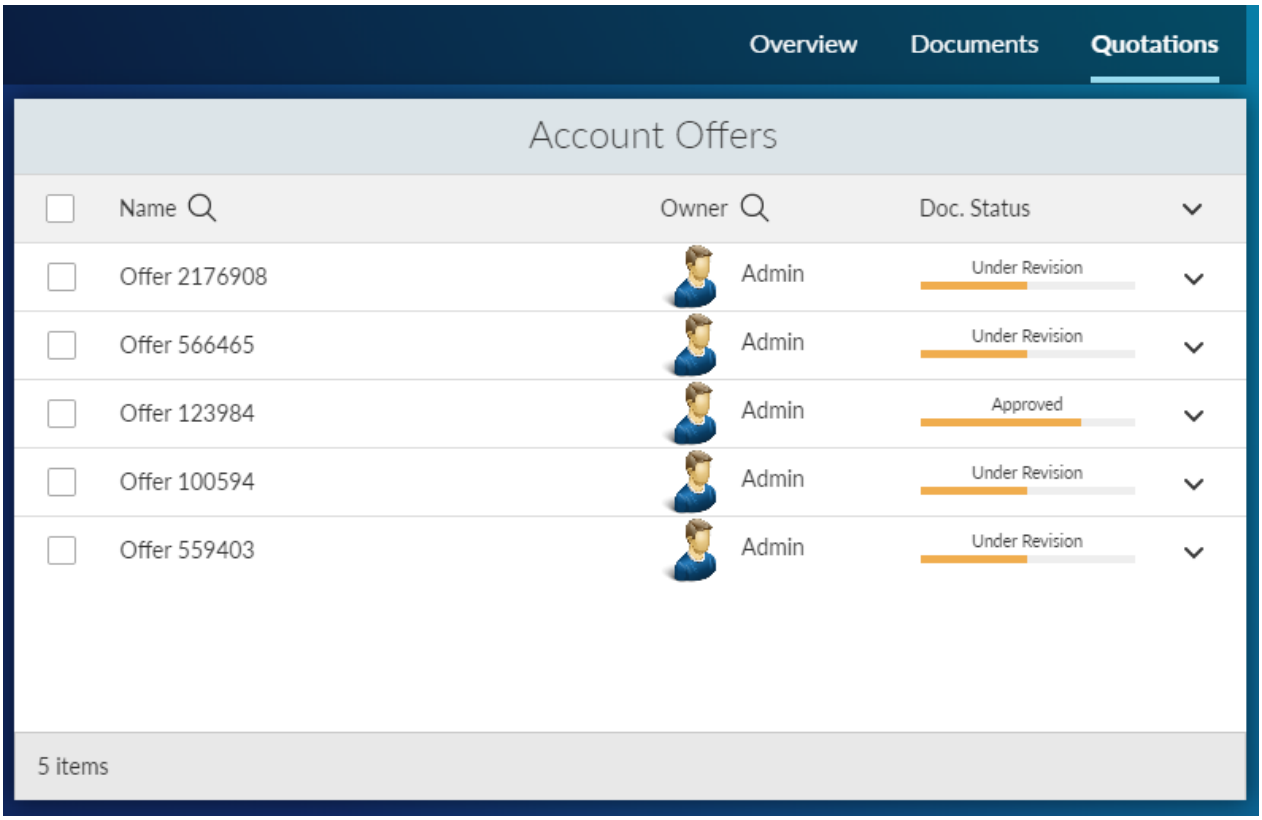
```
        function(_,Backbone, jQ, Radio){
            var amChannel = Radio.channel("ampagenotify");
            amChannel.on("printConsole", function(params){
                console.log("GOT request "+JSON.stringify(params));
            });
            amChannel.on("smartPage_action", function(action,param){
                console.log("GOT Page Action request. Action: "+action+ " parameter: "+param);
            });
            jQ("#filter").on("blur", function(){
                amChannel.trigger("updateTree",{'term':jQ(this).val()})
            })

        });
</script>"""
        ]
        ] )
    return
}

data =

    [
        [

            icon     : "csui-icon cs_vfolder", //mime_folder, cs_folder_root, cs_vfolder, cs_fol
            id       : 1,
            text     : "Roots",
            children :  [
                [
                    action   : "navigate", //Trigger a Smart View navigation
                    icon     : "csui-icon cs_folder_root", //cs_folder_root, cs_vfolder, cs_fold
                    id       : 2000, //The node will be used as the action's parameter
                    text     : "Home",
                    children : false
                ]

            ],
            state       : [
                opened    : true
            ]
        ],
        [

            action   : "printConsole", //Trigger a Tile action
            params   : "3", //This value will be passed to the script in a parameter named 'tile
            icon     : "csui-icon mime_folder",
            id       : 3,
            text     : "Folder (Lazy Loaded)",
            children : true,
            state       : [
                opened    : false
            ]
        ]
    ]


if(params.uiParentID == "3"){
    data[1].children = [
        [

            icon     : "csui-icon mime_folder",
            id       : 4,
            text     : "Sub Folder",
            children : [
                [
                    notify   : "smartPage",    //Triggers a Smart Page action noifying the provi
                    action   : "customAction", //The action to execute
                    params   : "2000",         //The action's parameter
                    icon     : "csui-icon mime_pdf",
```

```
                        id       : 5,
                        text     : "Notify Smart Page",
                        children : false
                    ],
                    [
                        action   : "printConsole",
                        params   : "2000",
                        icon     : "csui-icon mime_pdf",
                        id       : 6,
                        text     : "Execute Action",
                        children : false
                    ]

                ]
            ]

        ]
    }
    if(params.term){
        data = data.findAll{it.text.startsWith(params.term)}
    }
    json(data)
```

Content Script Node Table¶

The **Content Script Node Table** is an enhancement of the standard Node Table tile. The tile uses a Content Script as data source, allowing to set up any custom business logic to generate the list of nodes to be shown.

◉○
ExampleDatasource



```
def targetSpaceFilter = 2000


def subtypeFilter = "144".split(",")

if(params.widgetConfig){
    json([
    widgetConfig:[
        reloadCommands:[ "updateData" ],
        columnsWithSearch:[ "Owner", "Name" ]
    ]
    ])
    return
}

if(params.page?.contains("_") && params.page_list){
    if(params.page_list[0].contains("_") && !params.page_list?[1]?.contains("_")){
        params.page = params.page_list[1]
    }else if(!params.page_list[0].contains("_") && params.page_list?[1]?.contains("_")){
        params.page = params.page_list[0]
    }
}

def paging = [actual_count:0,
            limit:((params.limit?:"30") as int),
            page:((params.page?:"1") as int),
            page_total:0,
            range_max:0,
            range_min:0,
            total_count:0,
            total_row_count:0,
            total_source_count:0]

def pageSize = paging.limit
def offset = (paging.limit * (paging.page - 1))
def firstRow = offset + 1
def lastRow = firstRow + paging.limit

nodes = []

def nameFilter = null
if( params.where_name ){
    nameFilter = "%${params.where_name}%"
}

def ownerFilter = null
if( params.where_owner ){
    ownerFilter = "%${params.where_owner}%"
}

def sortingOrderParam     = 'desc'
def sortingColumnParam     = 'name'

def sortingOrder     = 'DESC'
def sortingColumn     = 'DTree.Name'


if( params.sort && params.sort.contains('_') ){

    def sorting = params.sort.split('_')

    sortingOrderParam     = sorting[0]
    sortingColumnParam     = sorting[1]

    sortingOrder = ( sortingOrderParam == 'asc' ) ? 'ASC' : 'DESC'


    switch( sortingColumnParam?.trim() ){
```

```
            case 'name' :
                sortingColumn = 'DTree.Name'
                break

            case 'owner' :
                sortingColumn = 'KUAF.ID'
                break

            default :
                sortingColumn = 'DTree.Name'
                break
        }

    }


    try{


        def queryParams = [targetSpaceFilter as String]
        def queryIndex = 1

        def permExpr = "(exists (select DataID from DTreeACL aclT where aclT.DataID=DTree.DataID and


        sqlCode = """ select DTree.DataID "DID",
                           DTree.Name "NAME",
                           COUNT(*) OVER() as "overall_count"

                       from DTree
                       LEFT JOIN KUAF ON DTree.UserID = KUAF.ID

                       where DTree.ParentID = %1 """

        if(subtypeFilter.size() == 1){
            sqlCode += " and DTree.SubType = %${++queryIndex} "
            queryParams << (subtypeFilter[0] as long)
        } else if( subtypeFilter.size() > 1 ) {
            sqlCode += " and DTree.SubType IN (${subtypeFilter.join(',')}) "
        }

        if(nameFilter){
            sqlCode += " and DTree.Name LIKE %${++queryIndex} "
            queryParams << (nameFilter as String)
        }

        if(ownerFilter){
            sqlCode += " and (KUAF.Name LIKE %${++queryIndex} OR KUAF.LastName LIKE %${queryIndex} )
            queryParams << (ownerFilter as String)
        }

        if(!users.current.canAdministerSystem){
            sqlCode += " and ${permExpr} "
        }

        sqlCode += """
                    ORDER BY ${sortingColumn} ${sortingOrder}
                        OFFSET ${offset} ROWS
                        FETCH NEXT ${pageSize} ROWS ONLY

                    """



        def queryResults

        if(queryParams){
            queryResults =  sql.runSQLFast(sqlCode, true, true, 100, *queryParams).rows
```

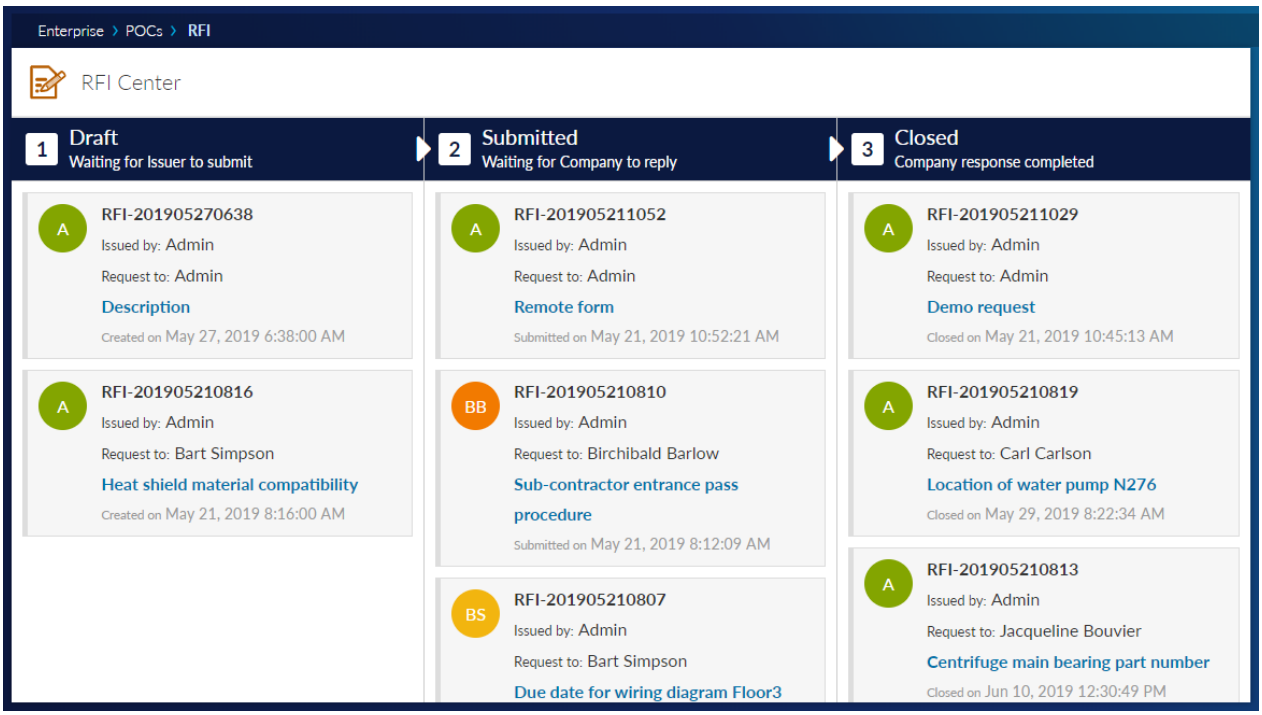
```
        } else {
            queryResults =  sql.runSQLFast(sqlCode, true, true, 100 ).rows
        }

        def totalCount = (queryResults) ? queryResults[0].overall_count : 0

        nodes = queryResults?.collect{it.DID as Long}



        paging << [
                actual_count:totalCount,
                page_total:((totalCount%paging.limit)+1),
                range_min:paging.page*paging.limit-paging.limit+1,
                range_max:(paging.limit*(paging.page+1)-totalCount)>0?(paging.limit*(paging.page+1)-
                total_count:totalCount,
                total_row_count:totalCount,
                total_source_count:totalCount]

    }catch(e){
        log.error("Error loading nodes table data",e)
        printError(e)
    }



    def drawStatusBar = { node ->

        def statusList = ['Draft', 'Under Revision', 'Approved', 'Published']
        def numSteps = statusList.size()
        def currStep = new Random().nextInt(statusList.size())
        def currStepName = statusList[currStep]


        def stepStyle = "height:100%; width:calc(100% / ${numSteps}); float:left; background-color:#

        def stepsHtml = ""

        (currStep + 1).times{
            stepsHtml += """<span style="${stepStyle}"></span>"""
        }


        return """
        <div style="text-align:center; font-size:.75em">${currStepName}</div>
        <div style="margin:3px 0; padding:0; height:5px; background-color:#eee;">${stepsHtml}</div>"
    }



    def slurper = new JsonSlurper()

    def processNode = { node, myNode ->

        /* Add your custom node post-processing here */

        //def myNode = asCSNode(node?.data.properties.id as long)

        node.data.amcsproxy =  [
            columns: [:],
            commands:[]
        ]

        //Add custom column: node.data.amcsproxy.colums.sample_column = "My custom Value"

            def owner = myNode.createdBy
            def ownerBox = "<span><img src='/otcs/cs.exe/pulse/photos/userphoto/${owner.ID}/2000' st
            node.data.amcsproxy.columns.owner = ownerBox
```

```
    node.data.amcsproxy.columns.comment = myNode.comment
    node.data.amcsproxy.columns.statusBar = drawStatusBar( myNode )

    return node
}

results = []

def fields = JsonOutput.toJson( [
    'actions': [ 'fields': [] ],
    'properties': [ 'fields': [] ],
    'versions': [ 'fields': [] ],
    'amcsproxy': [ 'fields': [] ],
])

//Identifies actions to be displayed for every node
//Node actions are return together with data request thay may lead to additinal response time
// [] - docman.getNodesRestV2JSon will not process actions.
//      Actions will be processed on a separate call based on the list provided (see returned js
// null - default list of actions will be returned
// ['open','properties','copy','move','edit'] - sample list of actions
// To ideal actions processing requires you to assign an empty list (see below) to the nodesActi
// using the 'actions' list property of the json object returned by this script (see last line)
def nodesActions = []

if( nodes.size() > 1 ){
    log.error("Nodes ${nodes}")
    temp = slurper.parseText( docman.getNodesRestV2JSon(nodes, fields, '{"properties":{"fields":
    theNodes = docman.getNodesFastWith(nodes, [], params, false, false, false)
    nodes.each{ node ->

        def jsonNode = temp.find{ it.data.properties.id == node }
        results << processNode(jsonNode, theNodes.find{it.ID == node}  )
    }


} else if (nodes.size() == 1 ){

    it = slurper.parseText(docman.getNodesRestV2JSon(nodes, fields, '{"properties":{"fields":["p
    processNode(it, docman.getNodeFast(nodes[0]))

    results = [it]
}




def columns = [

    type: [
            key:"type",
            name:"Type",
            type:2,
            type_name:"Integer",
            sort:false
        ]

    ,name: [
            key:"name",
            name:"Name",
            type:-1,
            type_name:"String",
            sort:true,
            align:"left"
        ]

    ,owner: [
            key:"owner",
            name:"Owner",
```

```
                type:43200,
                type_name:"String",
                sort:true,
                align:"left"
            ]

        ,statusBar: [
                key:"statusBar",
                name:"Doc. Status",
                type:43200,
                type_name:"String",
                sort:false,
                align:"left"
            ]

        ,comment: [
                key:"comment",
                name:"Comment",
                type:-1,
                type_name:"String",
                sort:false,
                align:"left"
            ]
    ]


    // actions - list of commands defined for all the nodes listed in the page
    // action=[] - will return all possible actions for a node
    json(
        [
        paging:paging,
        columnsWithSearch:[ "name" , "owner" ],
        results:results,
        columns:columns,
        tableColumns:columns,
        widgetConfig:[
            reloadCommands:[ "updateData" ]
        ],
        actions: ['open','properties','copy']
    ]
    )
```

Content Script Result¶

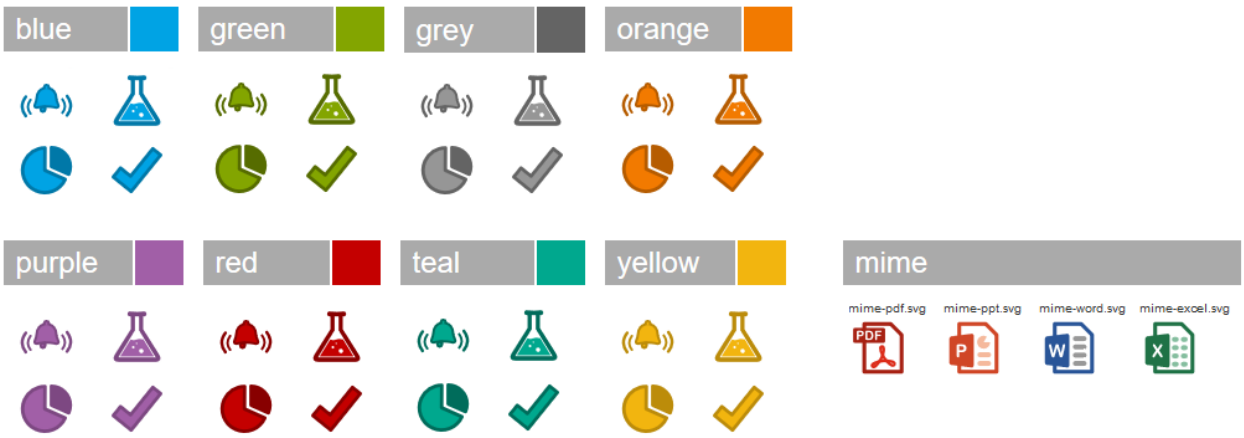
The **Content Script Result** is a general-purpose tile that can be used to inject any output generated by a Content Script Data source or a Smart Page into a SmartUI perspective.



Icon Reference Cheat Sheet¶

Iconset Color Codes¶

Module Suite icons are available in the following colors:



All Icons¶

A complete list of the currently available icons is shown below:

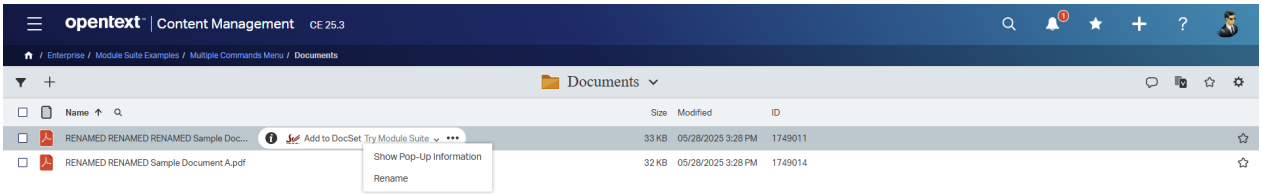
Next Steps¶

- Learn about Smart Pages Object creation and management
- Explore Smart View Overrides for customizing menus and columns
- Understand Tile Communication between different tiles

Smart View Overrides¶

Overview¶

Smart View overrides allow you to customize several aspects of the Smart View without having to rely on the Smart View SDK and without the need to deploy new artifacts on Content Server servers. This low-coding approach enables rapid customization of Smart View features.



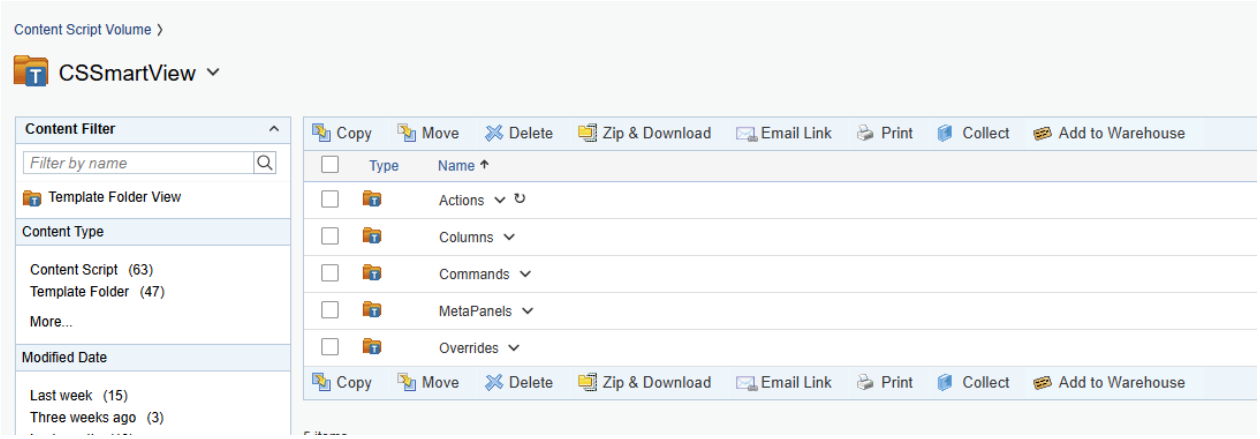
General Concepts¶

Like many other features in Module Suite, Smart View overrides follow a convention-based configuration approach. For applying a customization to the Smart View UI using one of the supported overrides, it is sufficient, in most cases, to create the appropriate script under the appropriate Content Script Volume folder.

Folder Structure¶

Smart View overrides are organized as follows:

```
Content Script Volume
└── CSSmartView
    ├── Actions      # Used to define lazy loaded actions to be displayed in nodes' related actionba
    ├── Commands     # Used to define new commands to be displayed in nodes' related actionbars
    ├── Columns      # Used to define custom dynamic columns to be displayed in Content Server space
    ├── MetaPanels   # Used to define custom dynamic Metadata Panels that can be added alongside an
    └── Overrides    # Overrides configuration. Its content determines when and where a particular o
```



Override Map (OM) and Actual Override Map (AOM)¶

Having a possible serious impact on the end user experience, it is important that the system is effective in calculating how, where and when overrides should be applied.

For this reason, Module Suite uses an elaborate algorithm to determine the **Actual Override Map (AOM)** to use when overrides should be applied.

Override Map Structure¶

The content of the **Overrides** folder is used to compute an Override Map (OM), specific to your repository, having the following structure:

```
OM = [
    "globals": [              // (1) Scripts to be always executed
        540588
    ],
    "type": [                 // (2) Scripts for specific subtypes
        "144": [              // (3) Scripts for Documents (subtype 144)
            548066
        ]
    ],
    "tenants": [              // (4) Tenant-specific overrides
        "497147": [            // (5) Tenant DataID
            "globals": [       // (6) Tenant global overrides
                548169
            ],
            "type": [          // (7) Tenant subtype overrides
                "144": [       // (8) Documents in this tenant
                    496932
                ]
            ],
            "ids": [           // (9) Specific node overrides
                "496931": [    // (10) Specific node DataID
                    545972
                ]
            ]
        ]
    ]
]
```

Override Evaluation Order¶

The configuration options are processed in the following order:

1. **Global Overrides** (scripts found directly under the Override folder) are executed first.
2. **Subtype Overrides** (scripts found within a Subtype folder - e.g. "S144" for Documents) will be evaluated second, and might overwrite the global overrides.
3. **Tenant Global Overrides** (any object in the tenant) will be evaluated next (3rd globally).
4. **Tenant Subtype Overrides** will go next (4th globally) to process objects of a specific subtype within the selected tenant.
5. **Tenant DataID Overrides** go last (5th globally). These are overrides targeting a specific object, and thus have the highest priority.

```
flowchart TD
    Start([Override Evaluation Starts]) --> Global[1. Global Overrides<br/>Override folder root]
    Global --> Subtype[2. Subtype Overrides<br/>S144, S0, etc.]
    Subtype --> TenantGlobal[3. Tenant Global Overrides<br/>D+TenantID/root]
    TenantGlobal --> TenantSubtype[4. Tenant Subtype Overrides<br/>D+TenantID/S+Subtype]
    TenantSubtype --> TenantDataID[5. Tenant DataID Overrides<br/>D+TenantID/D+NodeID]
    TenantDataID --> End([Final Override Map<br/>Highest Priority])

    style Global fill:#e1f5ff
    style Subtype fill:#fff4e1
    style TenantGlobal fill:#e8f5e9
    style TenantSubtype fill:#f3e5f5
    style TenantDataID fill:#ffebee
    style Start fill:#f5f5f5
    style End fill:#f5f5f5
```

How OM is Created¶

In order to determine the OM, the content of the **Overrides** folder is evaluated following this logic:

- **globals**: Contains the list of scripts stored directly under "Overrides"
- **type**: For each direct subfolder of "Overrides" that has a name starting with "S", an entry is created. The key is the target subtype (from the folder name), and the value is the list of scripts in that folder.
- **tenants**: For each direct subfolder of "Overrides" that has a name starting with "D", an entry is created. The key is the tenant's DataID, and the value is the tenant OM configuration.
- **ids**: Within tenant folders, subfolders starting with "D" are used to create entries in the "ids" map for specific node overrides.

Example Folder Structure¶

```
Overrides (ID: 00001)
├── GlobalScript (ID: 00002) - Content Script
├── S144 (ID: 00003) - AnsTemplateFolder
│   └── Document Script (ID: 00004) - Content Script
└── D1234 (ID: 00005) - AnsTemplateFolder (Tenant)
    ├── S0 (ID: 00006) - AnsTemplateFolder
    │   └── Folder Script (ID: 00007) - Content Script
    └── D5678 (ID: 00008) - AnsTemplateFolder
        └── Node Script (ID: 00009) - Content Script
```

This structure results in:

```
{
    "globals": [00002],
    "type": {
        "144": [00004]
    },
    "tenants": {
        "1234": {
            "globals": [],
            "type": {
                "0": [00007]
            },
            "ids": {
                "5678": [00009]
            }
        }
    }
}
```

Actual Override Map (AOM)¶

When a user changes the current space, the OM is evaluated by the framework against the users' permissions and the actual override map (AOM) associated to the space is determined.

AOM is determined by executing the relevant scripts in OM in the order described above. The AOM has the following form:

```
AOM = [
    "S144": [                    // (1) Commands/columns for subtype 144
        commands: ["comm_one", "comm_two", ...],  // (2) List of command keys
        columns: [                                // (3) Optional columns
            col_name: "col value",  // value can be HTML
            ...
        ]
    ],
    "D1234": [                   // (4) Commands/columns for specific node
        commands: ["comm_one", "comm_two", ...],
        columns: [
            col_name: "col value",
            ...
        ]
    ]
]
```

Where: - **(1)** represents commands and columns to be associated to all nodes having the identified subtype - **(3)** can be omitted if no custom columns are needed - **(4)** represents commands and columns to be associated to a specific node (identified by its id) - **(4)** takes precedence over **(1)**

Override Map Creation Timeline¶

Initial System Startup¶

1. **First User Access**: When any user first accesses a SmartUI application, the system begins the override definition loading process
2. **Definition Loading**: All Column, Command/Action, and Panel definitions are loaded from the Content Script volume
3. **User-Specific Caching**: If `amcs.amsui.volumeCache` is enabled, definitions are cached in memcached per user

Per-Space Navigation¶

1. **Lazy Initialization**: When users navigate to different OTCS folders/spaces, the Actual Override Map is built on-demand
2. **Permission Evaluation**: The system evaluates user permissions against Content Script volume objects
3. **Dynamic Computation**: AOM is computed based on current space context and user rights
4. **Script Execution**: Override scripts execute in the documented order hierarchy

Volume Cache Configuration¶

Parameter: `amcs.amsui.volumeCache`¶

- **Type**: Boolean
- **Default**: false
- **Scope**: Global system setting
- **Performance Impact**:
- **Enabled**: Significant performance improvement for override-heavy environments
- **Disabled**: Real-time computation on every space navigation (slower but always current)

Cache Storage Architecture¶

The system uses a sophisticated two-tier caching strategy:

1. **Memcached Layer**: Stores user-specific override definitions based on Content Script volume permissions
2. **Java Memory Layer**: Stores the Override Map (OM) structure

3. **Dynamic Computation**: Actual Override Map (AOM) generation on-demand during space navigation

Cache Management¶

- **Programmatic Clearing**: Use `amsui.clearCache()` API
- **Automatic Invalidation**: Cache invalidates on Content Script volume changes
- **User Isolation**: Each user maintains separate cached definitions

Direct Access Restrictions

Override Maps (OM) and Actual Override Maps (AOM) are internal system components and should not be accessed directly through custom code. Use the supported approaches: Custom Override Scripts, Override Configuration, Cache Management, and System Monitoring.

Smart View Custom Menus¶

Setting up a customization to the Smart View menu requires at least two components:

1. A **menu command definition script** (in `cssSmartView/Commands`)
2. An **override configuration script** (in `cssSmartView/Overrides`)

Menu Command Definition¶

Commands are defined in Content Scripts stored in the `cssSmartView/Commands` folder. These scripts return command definitions that specify how the command should appear and behave.

Basic Command Definition¶

```
return [
    [
        am: [
            exec: [
                mode: "script",
                script: 2644067,  // Content Script ID to execute
                params: [],
                refresh_on_success: true,
                on_success_action: "",
                newtab: false,
                url: ""
            ]
        ],
        baricon: null,
        icon: null,
        name: "My Custom Command",
        command_key: "my_custom_command",
        signature: "my_custom_command",
        scope: "multiple"  // single, multiple, or container
    ]
]
```

Command with Confirmation¶

```
return [
    [
        am: [
            confirmation: [
                required: true,
                title: "Confirm Action",
                message: "Are you sure you want to proceed?"
            ],
            exec: [
                mode: "script",
                script: 2644067,
                params: []
            ]
        ],
        name: "Delete Item",
        command_key: "delete_item",
        signature: "delete_item",
        scope: "single"
    ]
]
```

Command with Panel¶

```
return [
    [
        am: [
            panel: [
                width: 40,
                cssClass: "my-panel-class",
                slides: [
                    [
                        title: "My Panel",
                        script: 2644068  // Content Script for panel content
                    ]
                ]
            ],
            exec: [
                mode: "panel"
            ]
        ],
        name: "Open Panel",
        command_key: "open_panel",
        signature: "open_panel",
        scope: "single"
    ]
]
```

Grouped Commands¶

Commands can be grouped together in a flyout menu:

```
return [
    [
        // Parent command (flyout)
        am: [
            exec: [
                mode: "group"
```

```
            ]
        ],
        scope: "multiple",
        group: "info",
        flyout: "am_group",
        name: "Module Suite Actions",
        command_key: "am_group",
        signature: "am_group"
    ],
    [
        // Child command
        am: [
            exec: [
                mode: "script",
                script: 2644067,
                params: []
            ]
        ],
        name: "Content Script Action",
        command_key: "am_content_script",
        signature: "am_content_script",
        scope: "multiple",
        flyout: "am_group"  // References parent command
    ]
]
```

Override Configuration¶

Override configuration scripts determine when and where commands should be displayed. These scripts are stored in the `CSSmartView/Overrides` folder structure.

Override Map Format¶

For more control, return a map structure:

```
// Script in CSSmartView/Overrides
return [
    "D12345": [
        commands: ["x_selectDocument"]
    ],
    "D67890": [
        commands: ["x_selectDocument"]
    ]
]
```

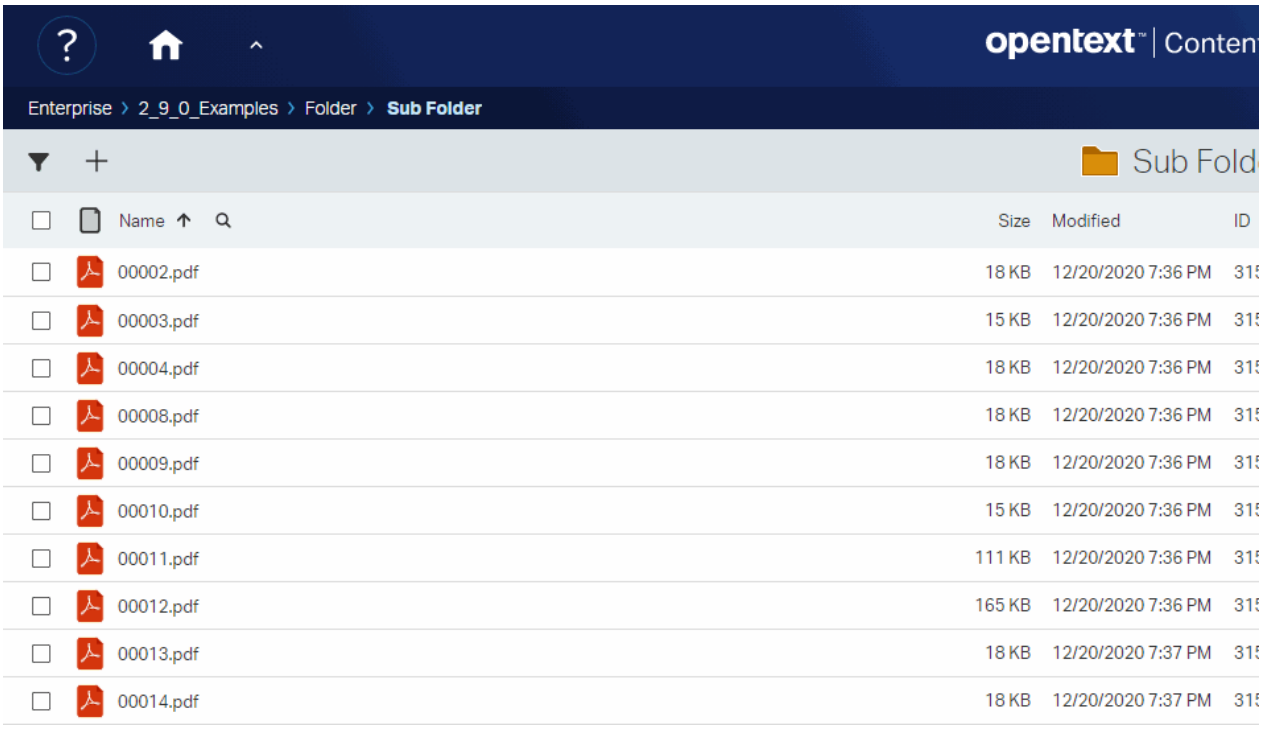
This approach allows multiple override behaviors to be defined in one script.

Dynamic Override Script¶

Override scripts receive a list of nodes in the execution context:

```
// Script receives 'nodes' variable in context
overrides = [:]

nodes.findAll { it.subtype == 144 }.each { node ->
    overrides["D${node.dataid}"] = [
        commands: ["training_override"]
```

```
    ]
}

return overrides
```



Smart View Custom MetaPanels¶

Setting up a customization to add custom metadata panels requires creating a **metadata panel definition script** in the `cs\SmartView/MetaPanels` folder. These panels can be displayed alongside an object's standard metadata panels, enabling enhanced or context-specific metadata display.

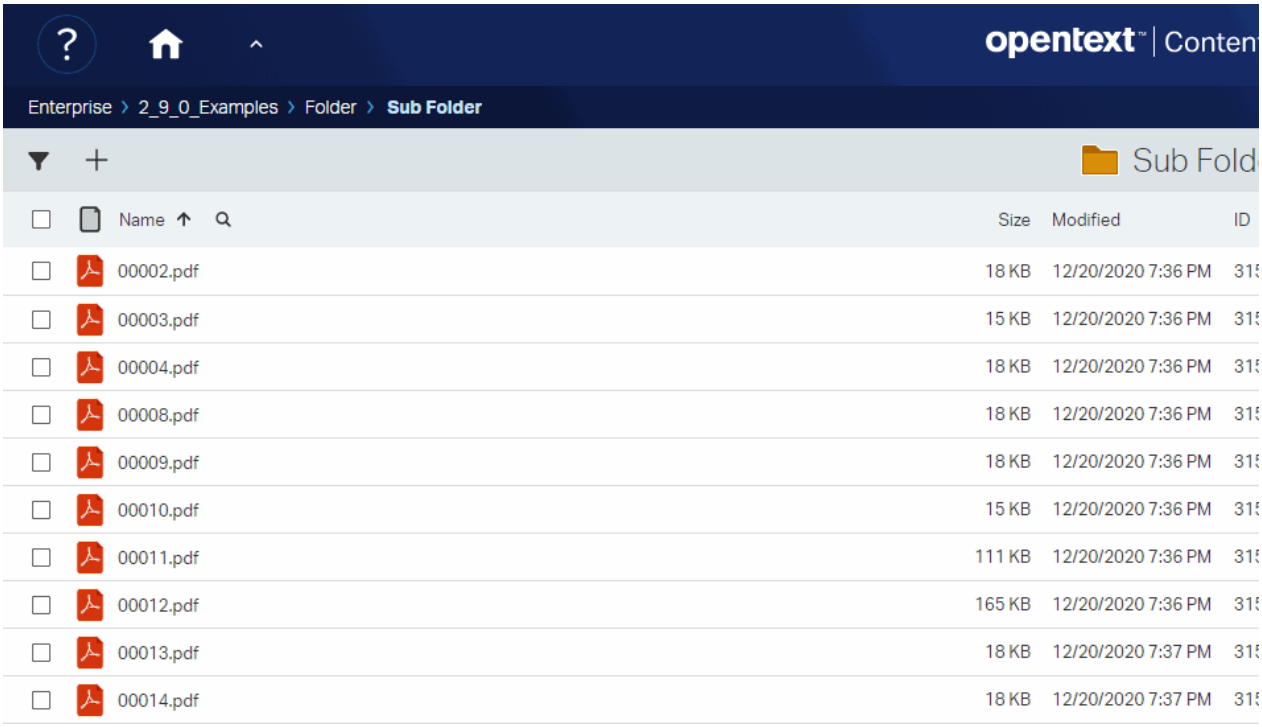
MetaPanel Definition Script¶

MetaPanel definition scripts return command definitions that specify how the metadata panel should appear and behave. The scripts can return a list of command definitions.

Basic MetaPanel Definition¶

```
// In CSSmartView/MetaPanels
// Commands scripts can return a list
return [
    [
        am: [
            exec: [
                mode: "group"  // (1) This command will act as our flyout
            ]
        ],
        scope: "multiple",
        group: "info",
        flyout: "am_group",  // (2) This command will act as our flyout
        baricon: null,
```

```
        icon: null,
        name: "Try Module Suite",
        command_key: "am_group",
        signature: "am_group"
    ],
    [
        am: [
            confirmation: [
                required: false,
                title: "",
                message: ""
            ],
            panel: [
                width: 40,
                cssClass: "",
                slides: [
                    [
                        title: "",
                        script: null
                    ]
                ]
            ],
            key: [
                code: 83,
                message: "",
                nogui: false
            ],
            exec: [
                mode: "script",
                script: 2644067,
                params: [],
                refresh_on_success: true,
                on_success_action: "",
                newtab: false,
                url: ""
            ]
        ],
        baricon: null,
        icon: null,
        name: "Content Script",
        command_key: "am_content_script",
        signature: "am_content_script",
        scope: "multiple",
        flyout: "am_group",
        selfBlockOnly: false
    ]
]
```



Smart View Custom Columns¶

Setting up a customization to add custom columns requires at least two components:

1. A **menu column definition script** (in `CSSmartView/Columns`)
2. An **override configuration script** (in `CSSmartView/Overrides`)

Column Definition Script¶

The column definition Content Script is responsible for filtering the list of columns available in a certain browse view, potentially removing available columns or injecting new ones.

The script is expected to return an object available within the context ("**nodesColumns**"), after processing it.

Basic Column Definition¶

```
// In CSSmartView/Columns
// Execution context includes:
// - nodesColumns: map associating node ids with their column definitions
// - nodes: list of node records
// - req: the original REST request record
// - envelope: the current REST API call envelope

def sampleColumn = [
    type: 43200,           // Should be 43200 for custom columns
    data_type: 43200,      // Should be 43200 for custom columns
    name: "Status",
    sort_key: "type",
    key: "sample_column"   // Unique identifier, must match override script
]
```

```
nodesColumns[3156087]?.add(sampleColumn)

// Must return the revised nodeColumns
return nodesColumns
```

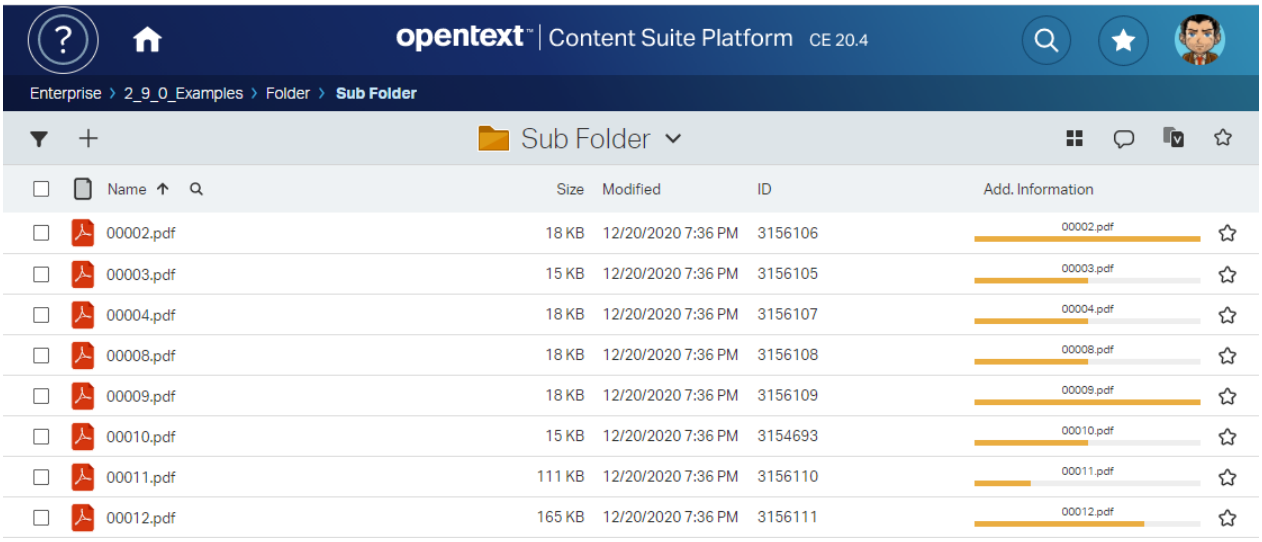
Column Definition Properties¶

Variable	Description
type	Should be 43200 for custom columns
data_type	Should be 43200 for custom columns
name	The plaintext value for the column header
sort_key	The identifier of the node value to be used for sorting this column
key	The unique identifier for this column. Must match the key provided in the override script

Column Override Definition¶

The override definition is shared with the custom menu overrides. In this case, we use the "columns" section:

```
// In CSSmartView/Overrides
retval = nodes.collect {
    [
        ("D${it.dataid}" as String): [
            commands: [],
            columns: [
                // Columns of type 43200 can be used to inject HTML
                sample_column: drawStatusBar(it),
                sample_icon_column: drawIcon(it)
            ]
        ]
    ]
}

return retval

// Helper function to generate column HTML
def drawStatusBar = { node ->
    def statusList = ['Draft', 'Under Revision', 'Approved', 'Published']
    def numSteps = statusList.size()
    def currStep = new Random().nextInt(statusList.size())
    def currStepName = statusList[currStep]

    def stepStyle = "height:100%; width:calc(100% / ${numSteps}); float:left; background-color:#F0AD
    def stepsHtml = ""

    (currStep + 1).times {
        stepsHtml += """<span style="${stepStyle}"></span>"""
    }

    return """
    <div style="text-align:center; font-size:.75em">${currStepName}</div>
    <div style="margin:3px 0; padding:0; height:5px; background-color:#eee;">${stepsHtml}</div>
    """
}
```



Smart View Custom Actions¶

Setting up a customization to add custom actions requires at least two components:

1. A **menu command definition script** (same as for menus, in `CSSmartView/Commands`)
2. An **action configuration script** (in `CSSmartView/Actions`)

Registering a Smart View Action¶

Registering an action differs from the registration of a standard menu entry as the customization is processed in a different, separate phase.

In this case, we use the **CSSmartView > Actions** folder. Here we create a Content Script that acts as a filter, whose aim is to process and manipulate an existing object called "actions" which is automatically injected in the context.

Actions Object Structure¶

The actions object is a mapping containing all the actions registered so far, for each object associated to the request, indexed by the object's DataID.

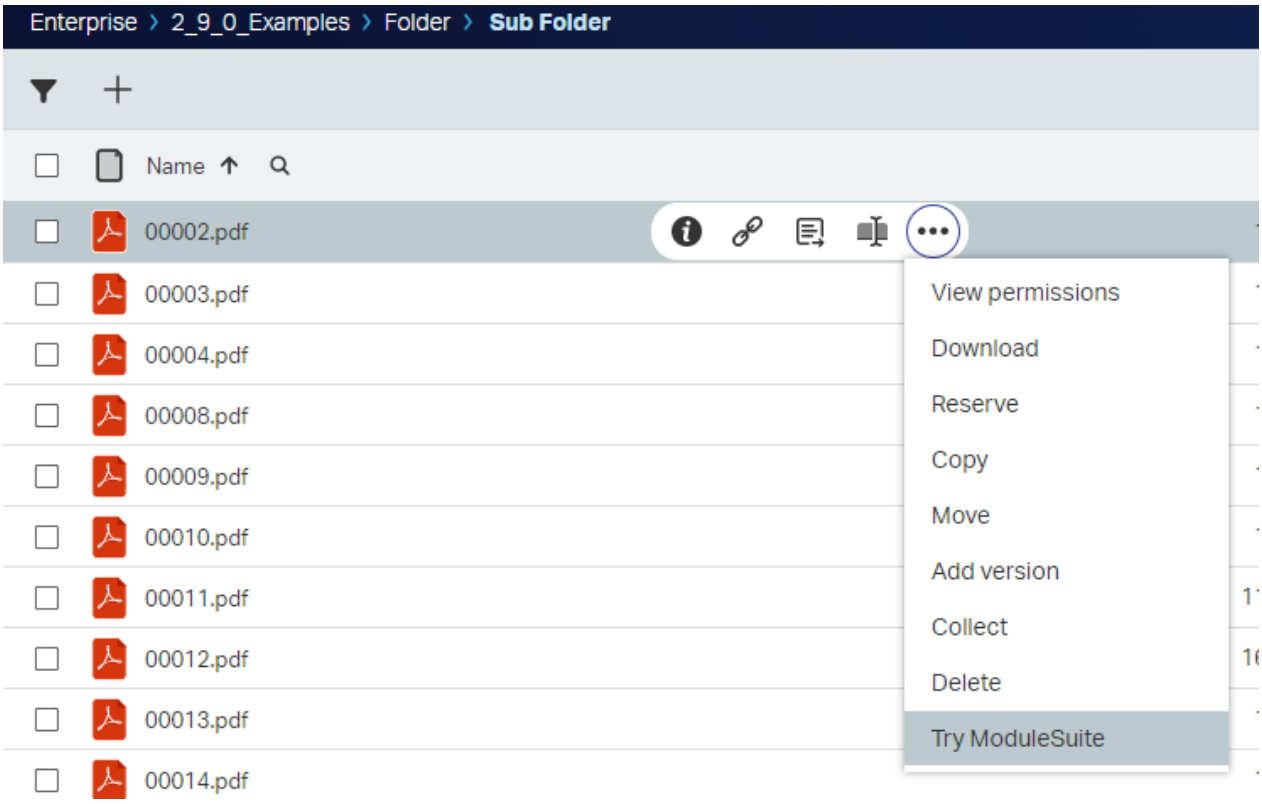
Example structure:

```
actions = [
    "656609": [
        data: [
            "Classify": [
                content_type: "application/x-www-form-urlencoded",
                method: "POST",
                name: "Add RM Classification",
                href: "/api/v2/nodes/2891606/rmclassifications",
                body: "{\"displayPrompt\":false,\"enabled\":false}",
                form_href: ""
            ],
            "zipanddownload": [
                content_type: "",
                method: "POST",
                name: "Zip and Download",
                href: "/api/v2/zipanddownload",
                body: "",
                form_href: ""
            ]
        ],
        map: [
            default_action: "open"
        ],
        order: [
            "Classify",
            "zipanddownload"
        ]
    ]
]
```

Action Registration Script¶

```
// In CSSmartView/Actions
// Execution context includes:
// - actions: map associating node ids with available actions
// - req: the current HTTP request
// - envelope: the REST API request's envelope

actions.each { action ->
    node = docman.getNodeFast(action.key as long)
    if (node.subtype == 144 && node.parentID == 973895) {
```

```
        // Add custom action based on business rule
        action.value.data["training_action"] = [
            body: "training_action"
        ]
        action.value.order.add("training_action")
    }
}
```



Command Execution and Return Values¶

Content Script scripts executed as commands can return execution information to the caller.

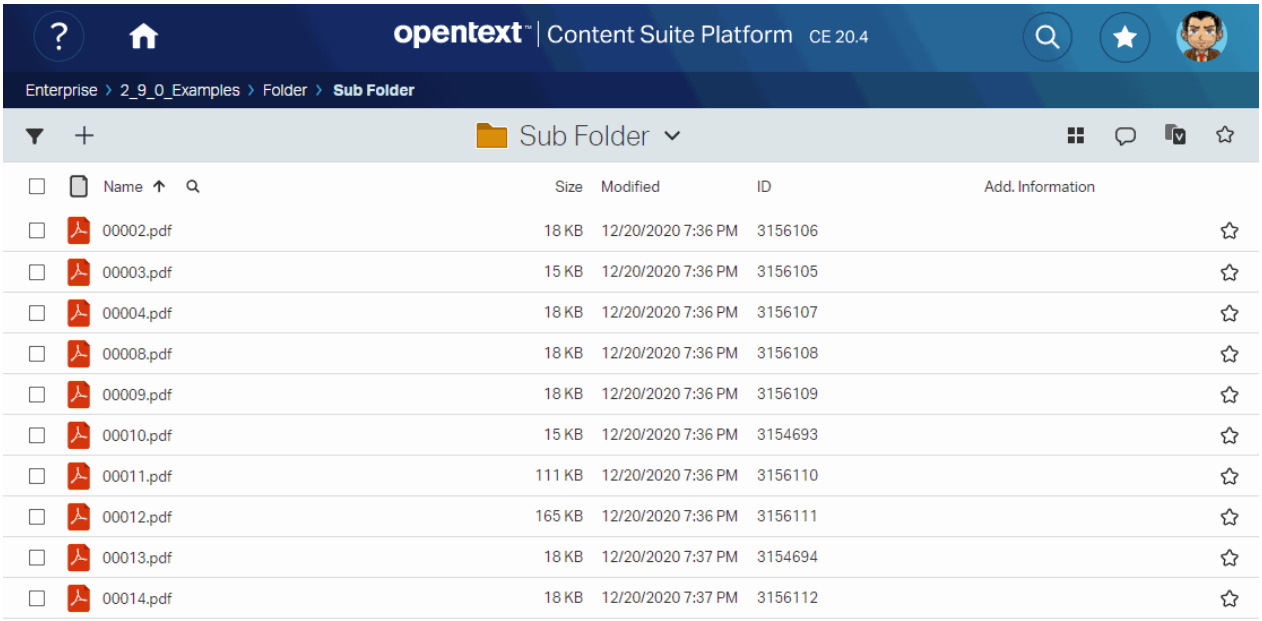
Success Message¶

```
// After script execution completes successfully
json([
    message: [
        type: 'success',
        text: "Operation completed successfully",
        details: "The document has been processed and saved to the repository."
    ]
])
```

Error Message¶

```
// After script execution encounters an error
json([
    message: [
```

```
            type: 'error',
            text: "Operation failed",
            details: "Unable to process the document. Please try again."
        ]
    ])
```



Best Practices¶

Override Scripts¶

- **Keep scripts focused**: Each script should handle a specific override type
- **Use meaningful names**: Name scripts descriptively
- **Document business logic**: Add comments explaining override conditions
- **Test thoroughly**: Test overrides in different contexts and with different user permissions

Performance¶

- **Enable caching**: Use `amcs.amsui.volumeCache` in production
- **Optimize queries**: Minimize database queries in override scripts
- **Cache expensive operations**: Cache results of expensive computations
- **Limit scope**: Only apply overrides where necessary

Security¶

- **Validate permissions**: Always check user permissions before applying overrides
- **Sanitize output**: Escape HTML in column values
- **Validate input**: Validate any parameters passed to commands
- **Audit actions**: Log important actions for audit purposes

Next Steps¶

- Learn about Tile Communication for inter-tile interactions
- Explore WebForms Integration for embedding forms
- Review Smart Pages Object for creating custom pages

Communication Between Different Tiles¶

Overview¶

Smart Pages tiles can communicate with each other and with other Smart View components through a message bus system. This enables dynamic, interactive interfaces where tiles can react to user actions and update their content accordingly.

Tile Communication Overview

> **Placeholder for Image**
>
> Add a diagram showing tile communication architecture

Radio Channel Communication¶

All tile communication in Smart Pages uses the **Radio channel messaging system**. The primary channel used is `ampagenotify`, which acts as a centralized message bus for all tile interactions.

Initializing the Radio Channel¶

Before tiles can communicate, they need to initialize the Radio channel:

```
csui.onReady2([
    'csui/lib/jquery',
    'csui/lib/underscore',
    'csui/lib/radio'
], function(jQ, _, Radio) {
    // Get the page message bus
    var amChannel = Radio.channel('ampagenotify');

    // Now tiles can communicate through amChannel
});
```

Communication Patterns¶

Pattern 1: Command-Based Communication¶

Tiles can trigger commands that other tiles listen for. This is the most common pattern for tile communication.

Sending Commands¶

```
// In Tile A's data source widgetConfig
html: """
    <script>
        csui.onReady2(['csui/lib/radio'], function(Radio) {
            var amChannel = Radio.channel('ampagenotify');

            // Trigger a command
            amChannel.trigger("updateChart", [
                {name: "filter", value: "active"}
            ]);
        });
    </script>
"""
```

Receiving Commands¶

```
// In Tile B's data source widgetConfig
html: """
    <script>
        csui.onReady2(['csui/lib/radio'], function(Radio) {
            var amChannel = Radio.channel('ampagenotify');

            // Listen for command
            amChannel.on("updateChart", function(params) {
                // Process params and update tile
                console.log("Received updateChart command", params);
                // Reload tile data or update UI
            });
        });
    </script>
"""
```

Pattern 2: Request-Response Communication¶

Tiles can request data from other tiles using the request pattern.

Making Requests¶

```
// Request data from another tile
var chart = amChannel.request("ampages:myChart");
if (chart) {
    // Use the chart object
```

```
    chart.updateData(newData);
}
```

Providing Data¶

```
// Register data provider
amChannel.reply("ampages:myChart", function() {
    return myChartInstance;
});
```

Pattern 3: Event Broadcasting¶

Tiles can broadcast events that multiple tiles can listen to.

Broadcasting Events¶

```
// Broadcast an event
amChannel.trigger("tile:dataUpdated", {
    tileId: "chart_1",
    data: newData
});
```

Listening to Events¶

```
// Listen for events
amChannel.on("tile:dataUpdated", function(eventData) {
    if (eventData.tileId === "chart_1") {
        // React to chart update
        updateRelatedTile(eventData.data);
    }
});
```

Common Communication Scenarios¶

Scenario 1: Chart and Filter Tiles¶

A filter tile updates a chart tile when filters change.

Filter Tile (Sender)¶

```
// In filter tile's data source
if(params.widgetConfig){
    json(widgetConfig: [
        html: """
            <script>
                csui.onReady2(['csui/lib/jquery', 'csui/lib/radio'],
                    function($, Radio) {
```

```
                        var amChannel = Radio.channel('ampagenotify');

                        $('#filterInput').on('change', function() {
                            var filterValue = $(this).val();
                            amChannel.trigger("updateChart", [
                                {name: "filter", value: filterValue}
                            ]);
                        });
                    });
            </script>
            <input id="filterInput" type="text" placeholder="Filter...">
        """
    ])
}
```

Chart Tile (Receiver)¶

```
// In chart tile's data source
if(params.widgetConfig){
    json(widgetConfig: [
        reloadCommands: ["updateChart"],  // Tile will reload when this command is triggered
        html: """
            <script>
                csui.onReady2(['csui/lib/radio'], function(Radio) {
                    var amChannel = Radio.channel('ampagenotify');

                    amChannel.on("updateChart", function(params) {
                        // The tile will automatically reload
                        // because "updateChart" is in reloadCommands
                    });
                });
            </script>
        """
    ])
} else {
    // Generate chart data based on params.filter
    def filter = params.filter ?: "all"
    // ... generate chart data ...
    json([/* chart data */])
}
```

Chart and Filter Communication

> **Placeholder for Image**
>
> Add a screenshot showing a filter tile updating a chart tile

Scenario 2: Links Tile and Node Table¶

A links tile triggers navigation or actions that update a node table.

Links Tile (Sender)¶

```
// In links tile's data source
json([
    data: [
```

```
        links: [
            [
                name: "Show Active Items",
                url: "#",
                action: "notify",
                command: "updateNodeTable",
                params: "active"
            ],
            [
                name: "Show All Items",
                url: "#",
                action: "notify",
                command: "updateNodeTable",
                params: "all"
            ]
        ]
    ]
])
```

Node Table Tile (Receiver)¶

```
// In node table tile's data source
if(params.widgetConfig){
    json([
        widgetConfig: [
            reloadCommands: ["updateNodeTable"]
        ]
    ])
} else {
    // Generate node list based on params.filter
    def filter = params.tile ?: "all"
    // ... query nodes based on filter ...
    json([/* node data */])
}
```

Scenario 3: Tree Tile and Content Display¶

A tree tile triggers content display in another tile when nodes are selected.

Tree Tile (Sender)¶

```
// In tree tile's data source
data = [
    [
        id: 2000,
        text: "Documents",
        action: "notify",
        params: "2000"
    ]
]

json(data)
```

The tree tile configuration should include:

```
if(params.widgetConfig){
    json([
        widgetConfig: [
            html: """
                <script>
                    csui.onReady2(['csui/lib/radio'], function(Radio) {
                        var amChannel = Radio.channel('ampagenotify');

                        // Tree will trigger "notify" action when node is clicked
                        // This will trigger the command specified in params
                    });
                </script>
            """
        ]
    ])
}
```

Content Tile (Receiver)¶

```
// In content display tile's data source
if(params.widgetConfig){
    json([
        widgetConfig: [
            reloadCommands: ["showNodeContent"]
        ]
    ])
} else {
    // Display content for the selected node
    def nodeId = params.tile ?: 2000
    def node = docman.getNodeFast(nodeId as long)
    // ... generate content display ...
    json([/* content data */])
}
```

Scenario 4: Smart Page Actions¶

Tiles can trigger Smart Page-level actions that affect multiple tiles.

Triggering Smart Page Actions¶

```
// In any tile's data source
json([
    data: [
        links: [
            [
                name: "Refresh All",
                url: "#",
                command: "smartPage",  // Notify Smart Page
                action: "refreshAll",   // Action name
                params: ""             // Action parameters
            ]
        ]
    ]
])
```

Smart Page Action Handler¶

```
// In Smart Page template or tile widgetConfig
html: """
    <script>
        csui.onReady2(['csui/lib/radio'], function(Radio) {
            var amChannel = Radio.channel('ampagenotify');

            amChannel.on("smartPage_action", function(action, param) {
                if (action === "refreshAll") {
                    // Trigger reload for multiple tiles
                    amChannel.trigger("updateChart", []);
                    amChannel.trigger("updateNodeTable", []);
                    amChannel.trigger("updateLinks", []);
                }
            });
        });
    </script>
"""
```

Reload Commands¶

The `reloadCommands` configuration option allows tiles to automatically reload when specific commands are triggered.

Configuration¶

```
if(params.widgetConfig){
    json([
        widgetConfig: [
            reloadCommands: [
                "updateChart",
                "refreshData",
                "filterChanged"
            ]
        ]
    ])
}
```

When any of these commands are triggered via `amChannel.trigger()`, the tile will automatically reload its data source.

Benefits¶

- **Automatic synchronization**: Tiles stay in sync without manual reload code
- **Simplified code**: No need to manually call reload functions
- **Consistent behavior**: All tiles use the same reload mechanism

Best Practices¶

Command Naming¶

- **Use descriptive names**: `updateChart` is better than `cmd1`
- **Follow conventions**: Use camelCase for command names
- **Namespace when needed**: Use prefixes like `tile:`, `page:`, `chart:` to avoid conflicts

Error Handling¶

```
amChannel.on("updateChart", function(params) {
    try {
        // Process command
    } catch (error) {
        console.error("Error processing updateChart:", error);
        // Show user-friendly error message
    }
});
```

Performance¶

- **Debounce rapid updates**: Use debouncing for frequently triggered commands
- **Batch updates**: Combine multiple updates into a single command when possible
- **Clean up listeners**: Remove event listeners when tiles are destroyed

Debugging¶

```
// Enable debug logging
amChannel.on("all", function(eventName, ...args) {
    console.log("Radio event:", eventName, args);
});
```

Advanced Patterns¶

Pattern: Observer Pattern¶

Multiple tiles observe a single data source tile:

```
// Data source tile broadcasts updates
amChannel.trigger("dataSource:updated", {
    data: newData,
    timestamp: Date.now()
});

// Multiple observer tiles listen
amChannel.on("dataSource:updated", function(eventData) {
```

```
    updateTile(eventData.data);
});
```

Pattern: Mediator Pattern¶

A central mediator tile coordinates communication between multiple tiles:

```
// Mediator tile
amChannel.on("tileA:action", function(data) {
    // Process and forward to tile B
    amChannel.trigger("tileB:update", processedData);
});

amChannel.on("tileB:response", function(data) {
    // Process and forward to tile C
    amChannel.trigger("tileC:update", processedData);
});
```

Next Steps¶

- Review Smart Pages Commands for detailed command reference
- Explore WebForms Integration for form-tile communication
- Learn about Smart View Overrides for custom menu and column interactions

commands javascript radio channel smartui

SmartPages commands

Integrate SmartUI Commands in your workflow¶

Introduction¶

SmartUI commands provide a powerful way to interact with the SmartUI framework through the Radio channel messaging system. These commands enable seamless communication between different components in SmartUI applications, allowing developers to create dynamic, interactive user interfaces without tight coupling between components.

Module Suite plays a crucial role in seamlessly incorporating SmartUI commands into your enterprise content management ecosystem. By leveraging the Radio channel messaging system, Module Suite empowers you to:

1. **Decouple components**: Enable loose coupling between UI components through event-driven communication, making applications more maintainable and extensible.

2. **Enhance user experience**: Display toolbars, viewers, side panels, and other UI elements dynamically based on user interactions and application state.
3. **Simplify integration**: Use a consistent command pattern across all SmartUI interactions, reducing complexity and learning curve.
4. **Enable dynamic content**: Load Smart Pages, Intelligent Viewing, and other content on-demand without full page refreshes.
5. **Improve responsiveness**: Show loading indicators and messages to provide feedback during asynchronous operations.
6. **Support flexible layouts**: Open side panels, display viewers, and manage multiple UI components simultaneously with precise control.

By integrating SmartUI commands through Module Suite, organizations can create rich, interactive user interfaces that enhance productivity, improve user experience, and provide seamless integration with Extended ECM capabilities.

> **SmartUI Commands Integration Considerations**
>
> While SmartUI commands offer significant benefits, it's important to consider factors such as command naming conventions, parameter validation, and error handling when integrating commands into your applications. Module Suite provides the necessary tools and patterns to address these considerations effectively.

Architecture and Communication¶

SmartUI commands use the Radio channel messaging system as a message bus for inter-component communication. This architecture enables decoupled, event-driven interactions between different parts of SmartUI applications.

Here's an overview of how the communication works:

```
flowchart TD
    subgraph App["SmartUI Application"]
        Component1[Component 1]
        Component2[Component 2]
        Component3[Component 3]
        Radio[Radio Channel<br/>ampagenotify]
    end

    Handler1[Toolbar Handler]
    Handler2[Viewer Handler]
    Handler3[Panel Handler]
    Handler4[Message Handler]

    Component1 -->|trigger command| Radio
```

```
Component2 -->|trigger command| Radio
Component3 -->|trigger command| Radio

Radio -->|route command| Handler1
Radio -->|route command| Handler2
Radio -->|route command| Handler3
Radio -->|route command| Handler4

Handler1 -->|update UI| Component1
Handler2 -->|update UI| Component2
Handler3 -->|update UI| Component3

style App fill:#f9f,stroke:#333,stroke-width:2px
style Radio fill:#bbf,stroke:#333,stroke-width:2px
style Handler1 fill:#bfb,stroke:#333,stroke-width:2px
style Handler2 fill:#bfb,stroke:#333,stroke-width:2px
style Handler3 fill:#bfb,stroke:#333,stroke-width:2px
style Handler4 fill:#bfb,stroke:#333,stroke-width:2px
```

Radio Channel Communication¶

All SmartUI commands are triggered through the **ampagenotify** Radio channel, which acts as a centralized message bus. This approach provides:

- **Decoupled communication**: Components don't need direct references to each other
- **Event-driven architecture**: Commands are triggered as events, enabling reactive programming patterns
- **Consistent interface**: All commands follow the same pattern, making them easy to learn and use
- **Extensibility**: New commands can be added without modifying existing components

Command Pattern¶

All SmartUI commands follow this consistent pattern:

```
amChannel.trigger("smartui:command:name", parameters);
```

Where: - `smartui:command:name` is the command identifier - `parameters` is an object containing the command-specific configuration

Typical Communication Sequence¶

Below is a diagram illustrating a typical communication sequence when using SmartUI commands:

```
sequenceDiagram
    participant User
    participant Component
    participant Radio as Radio Channel
    participant Handler
    participant UI

    User->>Component: User interaction
    Component->>Radio: trigger("smartui:command:name", params)
    Radio->>Handler: Route command to handler
    Handler->>Handler: Process command
    Handler->>UI: Update UI elements
    UI->>User: Display result
```

This pattern allows for clean separation of concerns, where components trigger commands without needing to know the implementation details of how those commands are handled.

Radio Channel Initialization

Before using any SmartUI commands, you need to initialize the Radio channel:

```
csui.onReady2(["csui/lib/radio"], function(Radio){
    var amChannel = Radio.channel('ampagenotify');
    // Now you can use amChannel.trigger() to send commands
});
```

Components of the SmartUI Commands Integration¶

Module Suite provides a comprehensive set of command handlers to enable seamless integration with SmartUI components. These handlers work together to offer a robust and flexible command-driven experience within the Extended ECM environment.

Command Handlers¶

SmartUI commands are processed by dedicated handlers that manage different aspects of the UI:

- **Toolbar Handler**: Manages action toolbars for nodes
- **Viewer Handler**: Handles Intelligent Viewing (IV) display
- **Smart Page Handler**: Manages Smart Page rendering
- **Panel Handler**: Controls side panel display and navigation
- **Loader Handler**: Manages loading indicators
- **Message Handler**: Displays global notifications

- **Search Handler**: Handles search interface navigation

```
graph TD
    A[SmartUI Commands] --> B[Command Handlers]
    B --> C[Toolbar Handler]
    B --> D[Viewer Handler]
    B --> E[Smart Page Handler]
    B --> F[Panel Handler]
    B --> G[Loader Handler]
    B --> H[Message Handler]
    B --> I[Search Handler]

    style A fill:#f9f,stroke:#333,stroke-width:2px
    style B fill:#bbf,stroke:#333,stroke-width:2px
```

Integration Use Cases¶

Module Suite offers various capabilities for integrating SmartUI commands into your Extended ECM environment. Let's explore common use cases and how to implement them.

Displaying Action Toolbars¶

Action toolbars allow users to interact with content nodes through a contextual menu of available actions. This functionality is valuable for implementing:

- Quick access to common actions (open, download, delete, etc.)
- Contextual action menus based on node type and permissions
- Customizable command sets for different use cases
- Inline action bars for table rows or list items

Example: Basic Toolbar Display¶

Here's a simple example of how to display an action toolbar:

Basic UsageAdvanced ConfigurationComments

```
csui.onReady2(["csui/lib/radio"], function(Radio){
    var amChannel = Radio.channel('ampagenotify');

    // Show toolbar for node 2000 with default container
    amChannel.trigger("smartui:show:toolbar", {
        id: 2000
        // el is optional, defaults to ".actions_2000"
    });
});
```

```
csui.onReady2(["csui/lib/radio"], function(Radio){
    var amChannel = Radio.channel('ampagenotify');

    // Show toolbar with custom container and specific commands
    amChannel.trigger("smartui:show:toolbar", {
        id: 2000,
        el: ".am-content-container",
        separateCommands: true,
        commands: ["open", "download", "delete"],
        blacklistedCommands: ["share"],
        delayRestCommands: false,
        text: {
            dropDownText: "More actions"
        },
        dropDownIconName: "csui_action_more32",
        addGroupSeparators: false,
        inlineBarStyle: "csui-table-actionbar-bubble",
        forceInlineBarOnClick: false,
        showInlineBarOnHover: true,
        error: {
            type: "error",
            text: "Unable to load toolbar",
            details: ""
        }
    });
});
```

The first example demonstrates the simplest usage, where only the `id` parameter is required. The toolbar will be displayed in the default container `.actions_{id}`.

The second example shows a more advanced configuration with: - Custom container selector - Specific commands to display - Blacklisted commands to exclude - Custom styling and behavior options - Error handling configuration

Command Filtering

Use `commands` to specify exactly which actions to show, or use `blacklistedCommands` to exclude specific actions from the default command set.

Displaying Document Viewers¶

Intelligent Viewing (IV) allows users to view documents directly within the application without leaving the current context. This functionality is valuable for implementing:

- Inline document preview
- Document comparison (multiple versions)
- Seamless document viewing experience
- Integration with Smart Pages for empty states

Example: Document Viewer Display¶

Single DocumentComparison ViewWith Smart Page Empty ViewComments

```
csui.onReady2(["csui/lib/radio"], function(Radio){
    var amChannel = Radio.channel('ampagenotify');

    // Show Intelligent Viewing for a single node
    amChannel.trigger("smartui:show:iv", {
        id: "docpreview_1",
        target: ".am-content-container",
        node: 2000,
        css: {
            height: "100%",
            position: "relative"
        },
        html: "",
        error: {
            type: "error",
            text: "Unable to load viewer",
            details: ""
        }
    });
});
```

```
csui.onReady2(["csui/lib/radio"], function(Radio){
    var amChannel = Radio.channel('ampagenotify');

    // Show comparison view for multiple versions
    amChannel.trigger("smartui:show:iv", {
        id: "docpreview_compare",
        target: ".am-content-container",
        node: [
            {id: 2000, version_number: 1},
            {id: 2000, version_number: 2}
        ],
        mode: "compare",
        css: {
            height: "100%",
            position: "relative"
        },
        error: {
            type: "error",
            text: "Unable to load viewer",
            details: ""
        }
    });
});
```

```
csui.onReady2(["csui/lib/radio"], function(Radio){
    var amChannel = Radio.channel('ampagenotify');

    // Show viewer with Smart Page as empty view
    amChannel.trigger("smartui:show:iv", {
        id: "docpreview_1",
        target: ".am-content-container",
        node: 2000,
        source: {
            source: 123,
            parameters: []
        },
        css: {
            height: "100%",
            position: "relative"
        },
        error: {
            type: "error",
            text: "Unable to load viewer",
```

```
            details: ""
        }
    });
});
```

The first example shows the basic usage for displaying a single document. The viewer will show the document and when closed, will display the empty HTML template.

The second example demonstrates the comparison view mode, which is useful for comparing different versions of the same document. When multiple nodes with the same ID are provided, the system automatically uses `version_number` as the identifier.

The third example shows how to use a Smart Page as the empty view. When the viewer is closed, instead of showing HTML, it will load and display the Smart Page specified in the `source` parameter.

Empty View Options

You can use either `html` (for static HTML templates) or `source` (for dynamic Smart Page content) as the empty view. If neither is provided, the container will be empty when the viewer is closed.

Node Parameter Flexibility

The `node` parameter accepts: - A single number: `node: 2000` - An array of numbers: `node: [2000, 2001, 2002]` - An array of objects: `node: [{id: 2000, version_number: 1}, {id: 2000, version_number: 2}]`

Displaying Smart Pages¶

Smart Pages can be loaded dynamically into containers, enabling dynamic content rendering based on server-side processing. This functionality is valuable for implementing:

- Dynamic content loading without page refresh
- Context-aware content display
- Reusable Smart Page components
- Flexible content rendering

Example: Loading Smart Pages¶

●○○

Using dataSourceUsing HTML TemplateComments

```
csui.onReady2(["csui/lib/radio"], function(Radio){
    var amChannel = Radio.channel('ampagenotify');

    // Show Smart Page with ID 2467915
    amChannel.trigger("smartui:show:smartpage", {
        id: "smartpage_1",
        target: ".am-content-container",
        css: {
            height: "100%"
```

```
        },
        dataSource: {
            source: 2467915,
            parameters: [
                {name: "target_folder", value: "2000"}
            ]
        },
        error: {
            type: "error",
            text: "Unable to load smart page",
            details: ""
        }
    });
});
```

```
csui.onReady2(["csui/lib/radio"], function(Radio){
    var amChannel = Radio.channel('ampagenotify');

    // Show custom HTML content
    amChannel.trigger("smartui:show:smartpage", {
        id: "custom_html_1",
        target: ".am-content-container",
        css: {
            height: "100%"
        },
        html: "<div><h1>Custom Content</h1><p>This is custom HTML content.</p></div>"
    });
});
```

The first example demonstrates loading a Smart Page script dynamically. The `dataSource` parameter specifies which Smart Page script to execute and what parameters to pass to it. This is useful when you want to display dynamic content that depends on server-side processing.

The second example shows how to display static HTML content directly. This is useful for simple content that doesn't require server-side processing or when you want to display custom HTML templates.

Mutually Exclusive Parameters

The `dataSource` and `html` parameters are mutually exclusive. You must provide either one or the other, but not both.

Displaying Side Panels¶

Side panels provide a slide-in interface for displaying additional content without navigating away from the main view. This functionality is valuable for implementing:

- Contextual information panels
- Search and filter interfaces
- Multi-step workflows
- Document viewing in side panels

Example: Side Panel Display¶

Basic Side PanelMultiple SlidesComments

```
csui.onReady2(["csui/lib/radio"], function(Radio){
    var amChannel = Radio.channel('ampagenotify');

    // Show side panel with Smart Page
    amChannel.trigger("smartui:show:sidepanel", {
        id: "myCustomPanelRight",  // Remember this ID to close the panel later
        panel: {
            width: 40,
            cssClass: "am_panel_class",
            layout: {
                header: true,
                footer: true,
                mask: false,
                resize: true
            },
            slides: [{
                title: "Search",
                subTitle: "",
                dataSource: {
                    source: 123,
                    parameters: [
                        {name: "am_ActionParams", value: "search"}
                    ]
                }
            }]
        }
    });
});
```

```
csui.onReady2(["csui/lib/radio"], function(Radio){
    var amChannel = Radio.channel('ampagenotify');

    // Show side panel with multiple slides
    amChannel.trigger("smartui:show:sidepanel", {
        id: "multiSlidePanel",
        panel: {
            width: 50,
            layout: {
                header: true,
                footer: true,
                resize: true
            },
            slides: [
                {
                    title: "Search",
                    subTitle: "Find documents",
                    dataSource: {
                        source: 123,
                        parameters: [{name: "am_ActionParams", value: "search"}]
                    }
                },
                {
                    title: "Filters",
                    subTitle: "Apply filters",
                    dataSource: {
                        source: 456,
                        parameters: [{name: "am_ActionParams", value: "filters"}]
                    }
                }
```

```
            ]
        }
    });
});
```

The first example shows a basic side panel with a single slide. The panel will slide in from the right side of the screen, occupying 40% of the viewport width.

The second example demonstrates a panel with multiple slides. Users can navigate between slides using the panel's navigation controls. Each slide can have its own Smart Page content.

Panel Width

The `width` parameter is specified as a percentage of the viewport width (vw). Common values are 30, 40, 50, or 100 for full-width panels.

Command Key

The `id` parameter is crucial as it serves as the **command_key** for closing the panel later. Make sure to use a unique identifier.

Displaying Loading Indicators¶

Loading indicators provide visual feedback during asynchronous operations, improving user experience by indicating that the system is processing a request.

Example: Loading Indicators¶

Manual LoaderAuto-hide Loader with AJAXComments

```
csui.onReady2(["csui/lib/radio"], function(Radio){
    var amChannel = Radio.channel('ampagenotify');

    // Show loader with custom label (must be manually hidden)
    amChannel.trigger("smartui:loader", {
        xhr: null,
        options: {
            label: "Loading documents..."
        }
    });

    // Hide loader manually after operation completes
    setTimeout(function() {
        // Use smartui:hide:loader if implemented
        // Or remove the loader element from DOM
    }, 5000);
});
```

```
csui.onReady2(["csui/lib/radio"], function(Radio){
    var amChannel = Radio.channel('ampagenotify');

    // Show loader tied to an AJAX request (auto-hides on completion)
```

```
    var xhr = $.ajax({
        url: "/api/data",
        method: "GET"
    });

    amChannel.trigger("smartui:loader", {
        xhr: xhr,
        options: {
            label: "Fetching data..."
        }
    });

    // Loader will automatically hide when AJAX request completes
    // (either success or error)
});
```

The first example shows a manual loader that must be explicitly hidden. This is useful when you need to control exactly when the loader disappears, such as after multiple asynchronous operations complete.

The second example demonstrates an auto-hiding loader. By providing a jQuery XHR object, the loader will automatically hide when the AJAX request completes, whether it succeeds or fails. This is the recommended approach for single AJAX operations.

> **Auto-hide Behavior**
>
> When you provide an `xhr` object, the loader uses the jQuery promise system to automatically hide when the request completes. This eliminates the need for manual cleanup code.

Displaying Messages¶

Global messages provide user feedback for operations, displaying success, error, warning, or informational notifications.

Example: Global Messages¶

Success and Error MessagesWarning and Info MessagesComments

```
csui.onReady2(["csui/lib/radio"], function(Radio){
    var amChannel = Radio.channel('ampagenotify');

    // Show success message
    amChannel.trigger("smartui:message", {
        type: "success",
        text: "Document saved successfully",
        details: "The document has been saved to the repository",
        options: null
    });

    // Show error message
    amChannel.trigger("smartui:message", {
        type: "error",
        text: "Operation failed",
        details: "Unable to save the document. Please try again.",
        options: null
```

```
    });
});
```

```
csui.onReady2(["csui/lib/radio"], function(Radio){
    var amChannel = Radio.channel('ampagenotify');

    // Show warning message
    amChannel.trigger("smartui:message", {
        type: "warning",
        text: "Warning",
        details: "This action cannot be undone",
        options: null
    });

    // Show info message
    amChannel.trigger("smartui:message", {
        type: "info",
        text: "Information",
        details: "Your session will expire in 5 minutes",
        options: null
    });
});
```

Messages are displayed globally and automatically dismiss after a timeout period. The `type` parameter determines the visual style and icon of the message:

- **success**: Green message with checkmark icon
- **error**: Red message with error icon
- **warning**: Yellow/orange message with warning icon
- **info**: Blue message with information icon

The `details` parameter provides additional context that can be expanded by the user.

> **Message Visibility**
>
> Messages are displayed in a non-intrusive manner and automatically disappear after a few seconds. Users can manually dismiss them if needed.

Closing Panels¶

Panels can be closed programmatically using specific close commands that match the panel's command key.

Example: Closing Panels¶

Closing Side PanelClosing IV Side PanelComments

```
csui.onReady2(["csui/lib/radio"], function(Radio){
    var amChannel = Radio.channel('ampagenotify');

    // Open a panel
    amChannel.trigger("smartui:show:sidepanel", {
```

```
        id: "myCustomPanelRight",  // This is the command_key
        panel: {
            width: 40,
            layout: { header: true, footer: true },
            slides: [{
                title: "Search",
                dataSource: {
                    source: 123,
                    parameters: []
                }
            }]
        }
    });

    // Close the panel using the command_key
    amChannel.trigger("smartui:dialog:myCustomPanelRight:hide");
});
```

```
csui.onReady2(["csui/lib/radio"], function(Radio){
    var amChannel = Radio.channel('ampagenotify');

    // Open IV side panel
    amChannel.trigger("smartui:show:iv:sidepanel", {
        id: "docpreview_panel",  // This is the command_key
        node: 2000,
        panel: {
            width: 100,
            layout: {
                header: false,
                footer: false,
                resize: true
            }
        }
    });

    // Close the IV side panel using the command_key
    amChannel.trigger("smartui:hide:docpreview_panel:iv:sidepanel");
});
```

The first example shows how to close a side panel opened with `smartui:show:sidepanel`. The command pattern is `smartui:dialog:{command_key}:hide`, where `{command_key}` matches the `id` used when opening the panel.

The second example demonstrates closing an IV side panel opened with `smartui:show:iv:sidepanel`. The command pattern is `smartui:hide:{command_key}:iv:sidepanel`.

> **Command Key Matching**
>
> The `command_key` in the hide command must **exactly match** the `id` used when opening the panel. If they don't match, the panel won't close.

Using Commands from Tiles Widgets¶

SmartUI commands can also be triggered from Tiles Widgets in Smart Pages, enabling command-driven interactions directly from tile components.

Example: Tile Widget Command¶

```
[
    size: 6,
    styleclass: "",
    command: "smartui:show:toolbar",      // Custom command name
    action: "notify",                       // Action type
    params: JsonOutput.toJson([            // Parameters as JSON
        id: 2000,
        el: ".am-content-container",
        separateCommands: true,
        commands: ["open", "download", "delete"],
        blacklistedCommands: ["share"],
        delayRestCommands: false,
        text: [
            dropDownText: "More actions"
        ],
        dropDownIconName: "csui_action_more32",
        addGroupSeparators: false,
        inlineBarStyle: "csui-table-actionbar-bubble",
        forceInlineBarOnClick: false,
        showInlineBarOnHover: true,
        error: [
            type: "error",
            text: "Unable to load toolbar",
            details: ""
        ]
    ]),
    newtab: false,
    type: 'blue',
    front: [
        body: "Show Toolbar",
        title: "Show Toolbar"
    ]
]
```

This example shows how to configure a tile widget to trigger a SmartUI command when clicked. The `command` parameter specifies the command name, `action` is set to "notify" to use the Radio channel, and `params` contains the command parameters as a JSON string.

Best Practices¶

When working with SmartUI commands, consider the following best practices:

Do: - Always use unique `id` values for each viewer/panel instance - Provide error handlers in your command parameters - Use the `xhr` parameter in `smartui:loader` for automatic loader dismissal - Use descriptive `command_key` values (e.g., "searchPanel" instead of "panel1") - Initialize the Radio channel before using commands - Handle errors gracefully with appropriate error configurations

Don't: - Don't use the same `id` for multiple instances - Don't hardcode node IDs - use variables or parameters - Don't use generic `command_key` values that might conflict - Don't forget to close panels when they're no longer needed - Don't ignore error handling - always provide error configurations

Command Organization

Organize your commands logically and use consistent naming conventions. This will make your code more maintainable and easier to debug.

Memory Management

Be mindful of opening multiple panels or viewers simultaneously. Consider closing unused instances to prevent memory leaks and performance issues.

Summary¶

This guide covered all the common SmartUI commands:

- **smartui:show:toolbar** - Display action toolbar
- **smartui:show:iv** - Display Intelligent Viewing in container
- **smartui:show:smartpage** - Display Smart Page in container
- **smartui:show:sidepanel** - Open side panel with Smart Page
- **smartui:show:iv:sidepanel** - Open side panel with Intelligent Viewing
- **smartui:loader** - Show loading indicator
- **smartui:message** - Show global message
- **smartui:search** - Open search interface
- **smartui:dialog:{command_key}:hide** - Close side panel
- **smartui:hide:{command_key}:iv:sidepanel** - Close IV side panel

All commands use the **ampagenotify** Radio channel for communication and follow consistent parameter patterns for easy integration into your SmartUI applications.

Script Console

Working with Script Console

Execution modes¶

Script Console is a runtime environment that features different execution modes (a shell, a script interpreter and a lightweight webserver) therefore it's the perfect solution when it comes to integrate Content Server with external systems. The simplest way to use the Script Console is to start it as a command line shell.

Script Console can run under both a Windows system and a Unix system, being based on a modular Java-based architecture. The main scripts for both the supported platforms are located under the "**bin**" directory in the runtime installation directory.

Command Line Shell Mode¶

In order to start the Script Console as a command line shell you have to execute the following command

```
>app-windows
```

Without any additional parameter. The system should respond you with the Script Console prompt (as shown in the figure below)

The prompt indicates the current system and its connection status. In the case of the figure above the current system has been labeled "TEST" and is currently off-line. New system can be added using the main configuration file of the Script Console. When newly installed a "TEST" system configuration is made available for future references.

An online help about the supported commands is available directly from the Script Console shell. Here below the list of all the commands available out-of-the-box:

loadcs

```
usage: loadcs -i 00000

Load a Content Script from a file or from Content Server

-e,--encoding <arg> The file encoding (platform default if not

specified)

-f,--file <arg> The local file to load as a script

-h,--help This help message

-i,--id <arg> The ID of the target script on the system
```

memsrc

```
usage: memsrc -g "MyGroup"

Search members

-@,--col-email COLUMN: Mail address

-a,--all Use a long listing format for results

-c,--match-contains MATCHING: Contains

-e,--match-endswith MATCHING: Ends with

-f,--col-first-name COLUMN: First name

-g,--filter-groups FILTER: Search only groups

-h,--help This help message

-k,--match-like MATCHING: Sounds like

-l,--col-last-name COLUMN: Last name

-m,--filter-members FILTER: Search any member
```

```
-n,--col-name COLUMN: Name

-s,--match-startswith MATCHING: Starts with

-u,--filter-users FILTER: Search only users
```

ls

```
usage: ls

List the children of the current node

-h,--help This help message

-l,--long Use a long listing format
```

rm

```
usage: rm "Node to delete" "Another node to delete"

Delete one or more nodes in the working node

-h,--help This help message

-i,--id Reference nodes by ID

-p,--parent <arg> Use specified parent in place of working node

-r,--regexp <arg> Match the node names to delete against the

specified regexp
```

mkdir

```
usage: mkdir "Folder Name"

Create a new folder

-h,--help This help message

-p,--parent <arg> The parent ID of the new folder
```

script

```
usage: script

Switch to scripting mode. In script mode you can write and save your one
script one line at a time

-h,--help This help message
```

quit | exit

```
Shutdown and exit
```

whoami

```
Information about the current user
```

loaddocs

```
usage: loaddocs -d /home/user/myDocs -i -r .\*.pdf

Load documents on Content Server

-d,--directory <arg> The local directory to load files from

-h,--help This help message

-i,--interactive Prompt for confirmation for each file

-n,--name Prompt for a new name for each file

-p,--parent <arg> The target directory

-s,--suffix <arg> Match the node names to delete with the

specified suffix
```

system

```
usage: system -options

List systems or switch the current system

-a,--add <arg> Add a new system
```

```
-c,--current The current system details

-h,--help This help message

-l,--list List all the available systems

-s,--system <arg> Switch to the target system
```

pwd

```
Print the current working node
```

mkuser

```
usage: mkuser bob -p passwd1 -g "MyGroup, Developers"

Create a new user

-a Public access enabled

-c Can create and update users

-g Can create and update groups

-h,--help This help message

-l Login enabled

-p,--password <arg> The initial password

-s Can administer system

-u Can administer users
```

interactive

```
Switch to interactive console mode. In interactive mode you can enter
Content Script commands and execute them directly.
```

sync

```
usage: sync

Synchronized console command scripts
```

```
-c,--commit Commit modified local scripts to Content Server

-h,--help This help message

-n,--name <arg> The single command to sync
```

su

```
usage: su bob

Impersonate a different user

-h,--help This help message

-r,--restore Restore the original logged in user
```

login

```
usage: login -options

Login to the specified system

-h,--help This help message

-i,--interactive Force credential prompt (useful id there are

saved credentials)

-k,--save Save the provided credentials (Crypted)

-p,--password <arg> The user's password

-s,--system <arg> The system to connect to

-u,--username <arg> The username
```

cd

```
usage: cd -i 2000

Change the current working node

-c,--category Switch to category WS
```

```
-e,--enterprise Switch to enterprise WS

-h,--help This help message

-i,--id <arg> The ID of the target node

-n,--nickname <arg> The nickname of the target node

-p,--personal Switch to personal WS
```

logout

```
Logout from the current system
```

loadConfig

```
usage: loadConfig -v -m Mode

Loads the current system Base Configuration in the Script Console
Configuration

-h,--help Usage Information

-m,--mode <arg> Mode: either BASE, CUSTOM, ALL

-v,--verbose Verbose
```

Creating new command

New commands can be registered using Content Script to implement them. Script Console comes with a set of example commands implemented through Content Scripts that a developer can use as a reference to create his own.

Script Interpreter Mode¶

The Script Console can also be executed as a Script interpreter (in order to execute a specific Content Script) in this case the Console should be executed specifying both the script to be executed and the system to log in:

```
>app-windows -c Script.cs -s SYSTEM
```

In order to be able to execute the Script Console with this Mode valid user's credentials should have been registered using the command:

```
login –k –i –s SYSTEM
```

Server Mode¶

A third way the Script Console can be executed is as a lightweight webserver. In this case the Console should be executed specifying both the port on which to listen for incoming connection and the system to log in:

In order to be able to execute the Script Console with this Mode valid user's credentials should have been registered using the command:

```
login –k –i –s SYSTEM
```

Script repositories¶

The Script Console organizes the registered Content Script in isolated repositories. A Script repository might be dedicated to a specific system (in this case the Scripts stored in this repository will be loaded and made available only when the user decides to login to that system), or to a specific extension.

Script Console extensions' script are made available through all the configured systems.

Script Console features a synchronization command (synch), that can be used, both when the Console is running as a shell as well as when the console is running as a web server, in order to synchronize a system repository with the contents of the corresponding **CSCommands** Template folder in the Content Script Volume of the current system.

Script Console Internal scheduler configuration file¶

The Script Console features an internal scheduler configurable through an XML configuration file (**cs-console-schedulerConfiguration.xml**) that is stored under the **config** directory.

The internal scheduler allows to plan and execute tasks to be automatically run in the Script Console. It is based on Quartz open source library (a well-known Java Scheduler). For further

information please make reference directly to the Quartz documentation http://quartz-scheduler.org/ *(http://quartz-scheduler.org/)* .

Scheduler disabled by default

The internal scheduler is disabled by default. To enable it you can either add the `-t` flag to your command line:

```
./app.sh -p 9090 -s TEST -t true
```

Or, if the console is executed as a web application, enable it using the **context-param** in your web.xml file:

```
...
<context-param>
    <param-name>app.timer</param-name>
    <param-value>true</param-value>
</context-param>
...
```

Extension for DocuSign

Working with DocuSign

This guide includes the basic set of operations that can be used to setup a document signing process using the **Module Suite Extension for DocuSign**.

Creating a signing Envelope¶

One of the core concepts when setting up a DocuSign signing process is the "**Envelope**", which represents the overall container for a transaction.

When defining an envelope, you will be able to provide all details of the transaction. The minimal set of information to provide includes:

- the **documents** to sign
- the **recipients** of the signing request
- the **message** they will receive

See the official DocuSign REST API guide *(https://developers.docusign.com/docs/esign-rest-api)* for more details on this topic.

The **docusign** Content Script service includes methods to programmatically create and send signing envelopes.

EXAMPLE: Creating a simple envelope¶

```
def contract      = docman.getDocument(123456)
String contractID = contract.ID as String

definition = docusign.getNewEnvelopeDefinition()
                     .setEmailSubject("XYZ contract for signature")
                     .setEmailBody("Please sign the contract.")
                     .addRecipient('signers', 'Homer J. Simpson', 'homer@example.com', 'Manager')
                     .addSignHereTab("homer@example.com", contractID, "Sign here", 1, 89, 100)
                     .addDocuments(contract)
                     .notifyOnEnvelopeCompleted()
                     .notifyOnEnvelopeDeclined()
                     .notifyOnEnvelopeVoided()

envelope = docusign.createEnvelopeAndSend(null, definition)

docusign.registerEnvelope(envelope) // This command will register the envelope locally on Content Se
```

EXAMPLE: Creating an envelope using a predefined template¶

When creating a new DocuSign envelope, it is possible to provide the envelope configuration in the form of a Map object. The structure of this map is compatible with the JSON format DocuSign uses to define Envelopes and Templates. For this reason, for complex envelope templates, a possible approach is to define the Template within your DocuSign account (using the visual editor to setup Recipients, Signing Tabs, etc.) and then export it and use it within your Content Script app.

```
def documentToSign      = docman.getDocument(123456)
def emailMessageSubject = "XYZ contract for signature"
def emailMessageBody    = "Please sign the contract."
def documentsToSign     = [documentToSign]

def user = users.current


def envDefinition = [

    "documents"         : documentsToSign,
    "emailSubject"      : emailMessageSubject,
    "emailBlurb"        : emailMessageBody,
    "signingLocation"   : "Online",
    "authoritativeCopy" : "false",
    "notification": [
        "reminders": [
            "reminderEnabled"   : "false",
            "reminderDelay"     : "0",
            "reminderFrequency" : "0"
        ],
        "expirations": [
            "expireEnabled" : "true",
            "expireAfter"   : "120",
            "expireWarn"    : "0"
        ]
    ],
    "enforceSignerVisibility" : "false",
    "enableWetSign"           : "true",
    "allowMarkup"             : "false",
    "allowReassign"           : "false",
    "messageLock"             : "false",
    "recipientsLock"          : "false",
    "recipients": [
        "signers": [ user ],

        /* Alternatively, a map structure can be provided to define recipients (required for externa

        "signers": [
            [
                "defaultRecipient"   : "false",
                "signInEachLocation" : "false",
                "name"  : "",
                "email" : "",
                "otuser":[
                          "name"  : user.displayName,
                          "email" : user.email,
                          "ID"    : user.ID
                        ],
                "accessCode"       : "",
                "requireIdLookup"  : "false",
                "routingOrder"     : "1",
                "note"             : "",
                "roleName"         : "Responder",
```

```
                "deliveryMethod"   : "email",
                "templateLocked"   : "false",
                "templateRequired" : "false",
                "inheritEmailNotificationConfiguration": "false",
                "tabs": [
                    //"signHereTabs": []
                ]
            ]
         ],
        */

        "agents"              : [],
        "editors"             : [],
        "intermediaries"      : [],
        "carbonCopies"        : [],
        "certifiedDeliveries" : [],
        "inPersonSigners"     : [],
        "recipientCount"      : "1"
    ],

    "envelopeIdStamping" : "true",
    "autoNavigation"     : "true"
]

def envDef = docusign.getNewEnvelopeDefinition(envDefinition)
                     .notifyOnEnvelopeSent()
                     .notifyOnRecipientCompleted()
                     .notifyOnEnvelopeCompleted()

def env =  docusign.createEnvelopeAndSend(null,  envDef)

envelope = docusign.registerEnvelope(env).envelope // Register this envelope on Content Server. This
```

Embedded recipients¶

Module Suite Extension for DocuSign supports **embedded signing** for authenticated OTCS users. When using this pattern, DocuSign delegates the task of identifying the recipients of the signing request to Content Server. Content Server is allowed to request the generation of a pre-signed **signing url**, which can be used by the recipient to sign the documents without having to authenticate with DocuSign. This approach avoids the context switching of the normal flow, which would require to open the system-generated email notification and access the DocuSign signing request from the provided link.

Refer to the official DocuSign REST API Guide - Embedding *(https://developers.docusign.com/docs/esign-rest-api/esign101/concepts/embedding/)* for further details on this topic.

When using the **embedded signing** pattern, recipients should be specified using a CSUser object.

EXAMPLE: Get a pre-authenticated signing URL for an OTCS internal user¶

In order to generate a signing URL for an **embedded recipient**, use the *docusign.getRecipientUrl(...)* API.

```
String envelopeID = 'xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx'
String username = 'Admin'


def user = users.getUserByLoginName('Admin')
def env  = docusign.getEnvelope(envelopeID)

String profile               = null
String recipientUserName     = user.name
String recipientEmail        = user.email
String recipientClientUserID = user.ID as String
String recipientID           = env.recipients.find{ it.clientUserID == recipientClientUserID }.recip

String nextUrl               = "http://mycontentserver.example.com/otcs/cs.exe"

String signingUrl            = docusign.getRecipientUrl( profile, envelopeID, recipientUserName, rec

redirect signingUrl
```

Envelope status update and signed document synch back¶

An important action to be performed when a signing workflow is concluded is to retrieve the signed documents and synchronize them back on your Content Server system. Module Suite Extension for DocuSign supports automating this task in different ways:

- Subscribe to DocuSign push notifications when the envelopes change state (webhook pattern)
- Poll the envelope status and update the local instance when a change is detected

The first approach (webhook) relies on the creation of an endpoint that can be invoked from DocuSign when changes happen. This pattern can be implemented by setting up the **Script Console DocuSign Extension**

The second approach (polling) can be implemented by using the *getEnvelopeUpdates(...)* API on the **docusign** service.

EXAMPLE: Poll DocuSign for Envelope updates and synch back documents¶

The following script can be scheduled to periodically update all active DocuSign envelopes.

Correct API usage

DocuSign monitors that the usage of the API is compliant with certain guidelines. Specifically, certain APIs cannot be invoked with a frequency that goes over a certain threshold. When scheduling polling scripts, make sure that the scheduling frequency complies with the DocuSign guidelines.

NOTE: This limitation can be overcome by using the **webhook** pattern, as described earlier.

```
res =  sql.runSQLFast("""SELECT AM_DocuSign.EnvelopeID ENVELOPEID
                         from AM_DocuSign where
                              AM_DocuSign.EnvelopeStatus not in ('completed', 'Completed')""", fals
if(res){
    docusign.getEnvelopesUpdates(null, res).each{

        docusign.updateEnvelope(docusign.getEnvelopeDetails(null, it.envelope))

        if(it.envelopeStatus == "completed"){
            docusign.getEnvelopeDocuments(null, it.envelope).each{ doc->
                doc.each{
                    if(it.key > 0){
                        docman.getNodeFast(it.key).addVersion(it.value)
                    }
                }
            }
        }
    }
}
```

How to

performances-tips

Content Script: Retrive information

Nodes¶

Getting Content Server nodes¶

All the objects stored on OpentText Content Server are referred as nodes in Content Script.

The base interface representing a node is the CSNode interface. CSNode is the base interface for most of the Content Script API objects *(/working/contentscript/scripts/#content-script-api-objects)*.

Almost all the Content Script API Objects inherit from CSNodeImpl which is the base-class implementing the CSNode interface. As said a *node* represents an object on Content Server.

Different Objects correspond to different implementation of the CSNode interface (e.g. Folders(*SubType=0*) are implemented by CSFolderImpl, Documents(*SubType=144*) correspond to CSDocumentImpl).

A CSNode (more generally speaking any Content Script API Object) features:

- **Properties**: this is information specific to the Content Server object (e.g. name, subtype, size, creation date) and may vary for each CSNode implementation. In order to be recognized as properties the CSNode fields must be decoretad with the `@ContentScriptAPIField`;
- **API Methdos**: these are the APIs used to manipulate and retrieve information associated with objects;
- **Features**: these are additional features that are not strictly related to objects (their are not object's properties) but depend on external factors: the way Content Server is configured (which modules are available, how are they configuration), on object's configuration, on the user's permissions on the objects, on the context in which the features are accessed etc.

The Content Script API service you are going to use the most for retriving nodes is the docman service.

docman features several methods that allows you to retrive a node given:

- its unique numeric identifier;
- its path
- its name and the container in which is located;
- its nickname;
- etc..

The Base API *(/working/contentscript/scripts/#api-services)* also features a asCSNode method that serves as a shortcut for the above mentioned use cases.

Performances-tip: Lazy loading

In order to optimize performances, Content Scripts lazy-loads information from OTCS 'database, which means that such information is not available until firstly accessed. docman APIs allow you to specify which information you want to load beforehand. Retriving the minimum amount of information necessary is tipically done using the APIs ending with the Fast suffix and is to be consider a **best practice** and might have a significant impact over your's application performances.

DoDon't

```
def node = docman.getNode(123456)

if((node.Invoice.Status as String) == "Paid"){ // The node is loaded with regular method at l
    ...
}
```

```
node = docman.getNode(123545)
if(node.parentID == -1 ){ //ParentID is a base property for CSNode and since we are only acce
    ...
}
```

Getting a node given its ID¶

```
def node = docman.getNode(2000, //NodeID (on most of the environments 2000 identifies the Enterprise
                      true, //'true' if Reference information shall be loaded
                      true, //'true' if Reservation information shall be loaeded
                      true, //'true' if Versions information shall be loaeded
                      true, //-true- if Current Version  shall be loaeded
                      true, //'true' if Node's features shall be loaeded
                      true, //'true' if Metadata  shall be loaeded
                      true, //'true' if Permissions information shall be loaeded
                     )

node = docman.getNode(2000) //this is a shortcut for docman.getNode(2000, true, true, false, false,

node = docman.getNodeFast(2000) //this is a shortcut for docman.getNode(2000, false, false, false, f

node = asCSNode(id:2000) //this is a shortcut for docman.getNode(2000)

node = asCSNode(2000)//this is a shortcut for asCSNode(id:2000)
```

Get a list of nodes given their IDs¶

```
docman.getNodesFastWith(
                        [2000L, 2006L], // List of nodes IDs
                        ["GIF", "promotedCmds", "defaultLink", "size", "tableName"], // List of addi
                        params, //Current request parameters
                        true, //'true' if Versions information shall be loaeded
                        true, //'true' if Node's features shall be loaeded
                        true //'true' if Permissions information shall be loaeded
                       )

docman.getNodesFast(2000L, 2006L) //this is a shortcut for docman.getNodesFastWith([2000L,2006L], []

docman.getNodes(2000L, 2006L) //this is a shortcut for docman.getNodesFastWith([2000L, 2006L], [], [
```

Get Volumes¶

The most common volumes can be easily accessed using a dedicated API featured by the docman service. If an API is not available a volume can be retrieved using a simple SQL query based on its subtype. Volumes come in handy when you want to retrieve a node by its path.

```
docman.getEnterpriseWS() //Enterprise Workspace

docman.getPersonalWS()   //Personal Workspace

docman.getCategoryWS()   //Category Workspace

docman.getContentScriptVolume() //Content Script Volume

/*
 161 -- Workflow Volume
 198 -- Classification Volume
 211 -- Reports Volume
 233 -- Database Lookups
 236 -- Database Connections
 274 -- Best Bets
 405 -- Recycle Bin
 541 -- Content Server Templates
 862 -- Connected Workspaces
 863 -- Workspace Types
*/

def node =  docman.getNodeFast(sql.runSQLFast("""Select "DataID"
                FROM DTree
                Where SubType = 161""", false, false, 0
             ).rows[0].DataID)
```

Get Nodes By Path¶

```
def ews = docman.getEnterpriseWS()

node = docman.getNodeByPath(ews, "Training:Folder")

node = docman.getNodeByPath( "Training:Folder") //this is a shortcut for docman.getNodeByPath(docman

node = asCSNode(path:"Training:Folder")//this is a shortcut for docman.getNodeByPath("Training:Folde
```

Performances-tip: Use the variable to avoid reloading the same information

In order to optimize performances, you should always assign information you know is not going to change (during your script execution) to Content Script variables so to avoid to reload them everytime they are accessed.

DoDon't

```
def ews = docman.getEnterpriseWS()

node = docman.getNodeByPath(ews, "Training:Folder")

node = docman.getNodeByPath(ews, "An:Other:Path")
```

```
node = docman.getNodeByPath( docman.getEnterpriseWS(), "Training:Folder")

node = docman.getNodeByPath( docman.getEnterpriseWS(), "An:Other:Path")
```

Users and Groups¶

Getting Content Server Users and Groups¶

Content Server Users and Groups are managed by the *users* service in the Content Script. *users* service operates with CSMember, CSUser and CSGroup classes. CSUser and CSGroup are the classes that are providing API to work with the Content Server Users and Groups correspondingly. CSMember is an abstract class for for CSUser and CSGroup objects. It is used in the API where both Users and Groups classes can be passed as a parameter or return as a method return value. *users* service provides set of methods to retrieve User or a Group:

- get current user (user who is actually executing Content Script)
- get user/group by id
- get group by name
- get user by login name
- list group members
- etc..

Get current User¶

```
def user = users.current // Will return CSUser object of the user that is executing the script
```

Get by member ID¶

```
CSMember member

// Pass User or Group by ID. Method will return CSUser or CSGroup class objects
```

```
//Pass ID of the Content Server User
member = users.getMemberById(1000)
out << member instanceof CSUserImpl // will display true

//Pass ID of the Content Server User
member = users.getMemberById(1001)
out << ( member instanceof CSGroupImpl ) // will return true

//Get User by ID
member = users.getUserById(1000) // will return CSUser class object

//Get group by ID
member = users.getGroupById(1001) // will return CSGroup class object
```

Get member by the name¶

```
CSMember member

//Get Member using User Login Name
member = users.getMemberByLoginName("Admin") // Will return CSUser class object

//Get Member using Group Name
member = users.getMemberByLoginName("DefaultGroup") // Will return CSGroup class object

//Get User by UserName
member = users.getUserByLoginName("Admin")

//Get Group by Name
member = users.getGroupByName("DefaultGroup")
```

Get members by ID¶

```
def members

//Get by IDs
members = users.getMembersByID(1000,1001)

//members[0] - is object of CSUser class
//members[1] - is object of CSGroup class
```

Permissions¶

Getting Content Server Node Permissions¶

Content Script *docman* service allows script developers to perform operations with the Content Server permissions model. To get get node permissions:

```
CSNode node = asCSNode(33561)
//Node permissions can be retrieved either
//calling CSNode getRigths() method
CSNodeRights nodeRights = node.getRights()
```

```
//or by calling docman method and passing node as an attribute
nodeRights = docman.getRights(node)
```

Content Server permissions model is represented as two classes *CSNodeRights* and *CSNodeRight*. *CSNodeRights* class contains all the permissions of the node. It's fields correspond to Content Server node permission type. **ownerRight** - Owner Permissions **ownerGroupRight** - Owner Group Permissions **publicRight** - Public Access Permissions **ACLRights** - list of Assigned permissions Every permission is an *CSNodeRight* object, with following fields: **rightID** - ID of the User/Group to whom this Right is assigned **permissions** - list of permissions set. Following options are possible:

```
1   [SEE, SEECONTENTS, MODIFY, EDITATTRIBUTES, RESERVE, ADDITEMS, DELETEVERSIONS, DELETE, EDITPERMIS
```

To get node permissions:

```
//To get Owner Permissions
out << nodeRights.ownerRight.permissions

//To get Assignemt Permissions Users with their permissions
def assignedAccessUsers = [:]

nodeRights.ACLRights.each{ right ->
    def currUser = users.getMemberById(right.rightID);
    assignedAccessUsers[currUser.name] = right.permissions
}

out << assignedAccessUsers
```

There are set of methods to check if current user has special permissions against the node. Methods to check permission are implemented for CSNode and they are prefixed with "has" and than following permissions description:

- hasAddItemPermission()
- hasDeletePermission()
- hasDeleteVersionsPermission()
- hasEditAttributesPermission()
- hasEditPermissionsPermission()
- hasModifyPermission()
- hasReservePermission()
- hasSeeContentsPermission()
- hasSeePermission()

Sample validation:

```
CSNode node = asCSNode(33561)

out << node.hasDeletePermission() //will return TRUE if current user has Delete pemissions on a node
```

Categories¶

Getting Node Categories¶

Content Script *docman* service allows to performs full set of actions related to Content Server categories. Below you will find samples how to get Category definition and get Content Server node categories along with its attribute values.

```
def category = docman.getCategory(self.parent, "User Info") // Object of type CSCategory

def attributesMap = category.getAttributes() // Get map with Category Attributes

def firstNameAttr = category.getAttribute(attributesMap[2 as Long]) // get definition of the attribu

out <<  "Attribute ${firstNameAttr.getDisplayName()} has default value set to: ${firstNameAttr.value
```

Get value of the category attributes applied to a node:

```
def node = docman.getNodeByName(self.parent, "Folder With Categoty")

//Get Attribute value
def attrValue = node."User Info"."First Name" as String
out << "The current value of First Name is now ${attrValue} <br/>"

//get first attribute value
attrValue = node."User Info".Phone
out << "Get first Phone attribute value ${attrValue} <br/>"

//get all attribute values
attrValue = node."User Info".Phone as List
out << "Get all Phone attribute values ${attrValue} <br/>"
```

You can always export the category as a map, and later on update it from the very same map:

```
out << node."User Info" as Map
```

Classification¶

Manipulation with a node Classifications in Content Script is performed by the *classification* service. This sections describes how to get classifications applied to a node.

First of all if you need to check if node is classifiable:

```
def node = docman.getNodeByName( self.parent, "Test Folder")

//Check if Classification can be applied to the node
out << "Classification can be applied to a node: ${classification.isClassifiable(node)}"
out << "<br>"

//List classifialbe subtypes
```

```
out << "Classification can be applied to following node subtypes:"
out << "<br>"
out << classification.listClassifiebleSubTypes()
```

To get classifications:

```
def node = docman.getNodeByName( self.parent, "Test Folder")

// get node classifications
def classifications = classification.getClassifications(node)

//Will return list of classifications applied to a node
out << classifications.collect { it.name }
```

Executing SQL queries¶

Content Script API allows execution of SQL statements against Content Server database, without the need for creatomg a LiveReport object. *sql* service has a set of methods allowing developer to run SQL queries.

Not all DBMS are equal

Please keep in mind DBMS server SQL specific syntax of the queries used. Adapt provided queries to the DBMS server type in your environment.

Execute a simple SQL query¶

```
out << sql.runSQL("""select * from DTree where %1 and ParentID = %2 and ModifyDate > %3""", //SQL Co
           true, // true if the query must be executed using a cursor
           true, // true if the query must be wrapped in a transaction (required administrative priv
           10,   // numer of records to be returned

                              // Below the list of optional parameters
           "#FilterObject:0", // Parameters can be a LiveReport query template expression
           2000,              // Integers
           1.year.ago).rows   // Dates
                              // Strings
```

The above query is executed with three parameters, specified as %N in the SQL statement.

SQL execution methods are returning *CSReportResult* class object. To get query executing result *rows* feature should be used, as in the example above.

Another option to run SQL queis utilization of the `sql.runSQLFast()` methods. Syntax for *"Fast"* methods is the same. These methods are faster implementation of the SQL execution script, but the compromise is that they are not ThreadSafe (i.e. not to be used in multi-threaded scripts).

Execute a SQL query with pagination¶

In some cases it is required to implement queries that return paginated data, e.g. for browsing pages. *sql* exposes a set of methods that allow developers to easily build such queries The example below provides an overview of the usage of `sql.runPaginatedSql()` API:

```
def sqlProjections = "DataID, Name"
def fromClause = "DTree dt"
def whereClause = "SubType = 0"
def pageSize = 5
def transaction = true

def runPaginatedQuery = { firstRow ->

    def sqlResult = sql.runPaginatedSql(sqlProjections, fromClause, whereClause, firstRow, pageSize,

    out << "<br>"
    out << "Start row ${firstRow}"
    sqlResult.rows.each { row ->
        out << "<br>"
        out << "Folder Name: ${row.name}. Name: ${row.dataid}"
    }
}

runPaginatedQuery(1)
runPaginatedQuery(6)
```

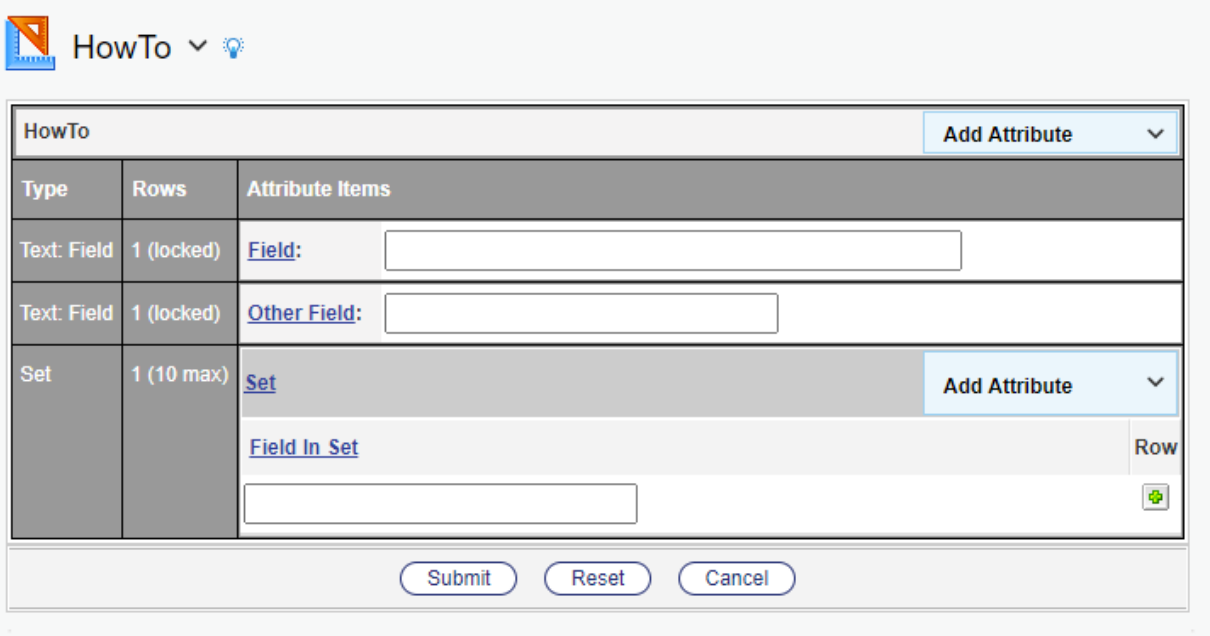
Working with Forms¶

Content Server Forms and Form Templates objects can be manipulated with Content Script through the *forms* service API.
The most important Service API Objects returned by the aformentioned service are: *CSForm*, *CSFormTemplate* and *Form*

While *CSForm* is used to manipulate the Content Server Forms objects (e.g. changing name, applying categories and classifications, changing permissions etc...) the *Form* type is used to represent the data submitted (record) through the form.

Objects used in this paragraph's examples

The examples presented in this paragraph are all making use of a Form Object named **HowTo Form** associated to a FormTemplate object named **HowTo** having the following structure.



The FormTemplate object has been configured to be associated to an SQL Table named **Z_HowTo**.
At the time of configuration Content Server produced the following SQL DDL instructions:

```
create table Z_HowTo
(
VolumeID bigint not null,
DataID bigint not null,
VersionNum bigint not null,
Seq bigint null,
RowSeqNum int default 1 not null,
IterationNum int default 1 not null,
Field nvarchar(255) null,
Other_Field nvarchar(255) null
)
/

create index Z_HowTo_Index1
on Z_HowTo ( VolumeID, DataID, VersionNum, Seq )
/

create table Z_HowToSet
(
VolumeID bigint not null,
DataID bigint not null,
VersionNum bigint not null,
Seq bigint null,
SubSeq int null,
RowSeqNum int default 1 not null,
IterationNum int default 1 not null,
Field_In_Set nvarchar(255) null
)
/

create index Z_HowToSet_Index1
on Z_HowToSet ( VolumeID, DataID, VersionNum, Seq )
/
```

Which once executed, resulted in the creation of two tables: **Z_HowTo** and **Z_HowToSet**

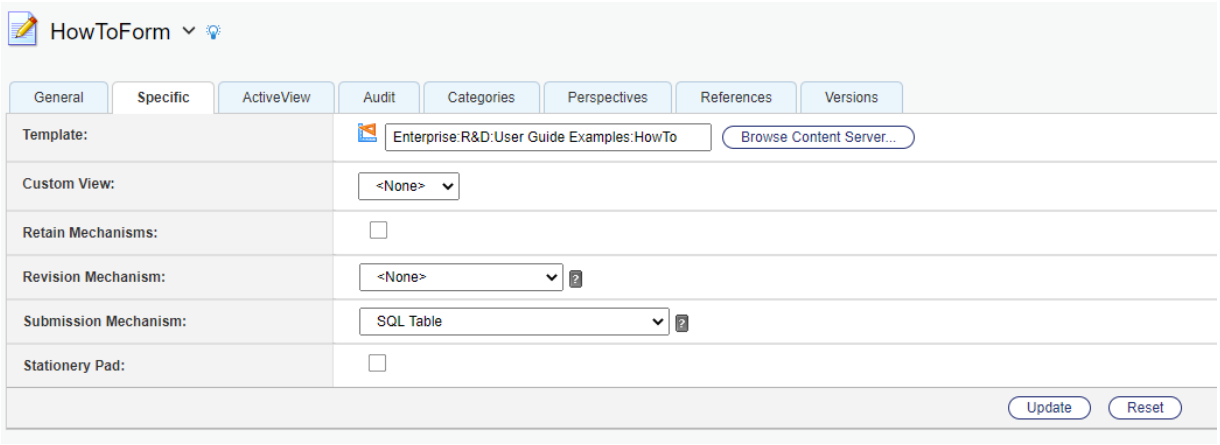
Z_HowTo (otcs)

Column Name	Data Type	Allow Nulls
VolumeID	bigint	☐
DataID	bigint	☐
VersionNum	bigint	☐
Seq	bigint	☑
RowSeqNum	int	☐
IterationNum	int	☐
Field	nvarchar(255)	☑
Other_Field	nvarchar(255)	☑
		☐

Z_HowToSet (otcs)

Column Name	Data Type	Allow Nulls
VolumeID	bigint	☐
DataID	bigint	☐
VersionNum	bigint	☐
Seq	bigint	☑
SubSeq	int	☑
RowSeqNum	int	☐
IterationNum	int	☐
Field_In_Set	nvarchar(255)	☑
		☐

The Form object uses, as a submission mechanism, the SQL Storage option, while no revision mechanism has been associated to it.



Retrive submitted data¶

To get the Content Server Form associated submitted data you can leverage the **listFormData*** APIs, these APIs accept an optional **filters** parameter, which can be used only for Forms having **SQL Table** as associated submission mechanism. Filters are Maps having as keys the names of the tables you want to filter data from and as values a valid SQL **where** clause:

ScriptOutput

```
def writer = new StringWriter()
def html = new MarkupBuilder(writer)
out<< template.evaluateTemplate("#csresource(['bootstrap'])")
html.table(class:"table"){
    thead{
        tr(class:"danger"){
            th("Field")
            th("Other Field")
            th("Set")
        }
    }
    tbody{
        formNode.listFormData(["Z_HowTo":" Seq in (select Seq from Z_HowToSet where Field_In_Se
            tr{
                td(form.field.value)
                td(form.otherField.value)
                td{
                    table(class:"table table-condensed"){
                        thead{
                            tr(class:"danger"){
                                th("Field in Set")
                            }
                        }
                        tbody{
                            form.set.each{ row->
                                tr{
                                    td(row.fieldInSet.value)
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}
out << writer.toString()
```

Field	Other Field	Set
one		Field in Set one
two		Field in Set two

```
def formNode = docman.getNodeByName(self.parent, "User Info Form") //returns a CSFormImpl node
def submittedData = formNode.listFormData()
```

In the script above *formNode* in CSForm object type that has API implemented to work with Content Server Forms. *submittedData* is a list of `Form` object types that corresponds to certain record of the submitted form data. To access fields of the form:

```
//List sumbitted data
//Access Form fields
submittedData.each {form ->
    out << "User ${form.firstName[0]} ${form.lastName as String}. Age ${form.age as String}"
    out << "<br>"
}
```

In the example above following form attributes are accessed:

Field Name	Normalized
First Name	firstName
Last Name	lastName
Age	age

In scripts, form field values can be accessed using the following notation *form.normalizedname.value*

where normalization is performed by the Content Suite Framework.

```
// Initalize form field values: some examples

form.wordsWithSpaces.value = “TEST VALUE E” // Form template field name: words with spaces

form.camelcase.value = “TEST VALUE D” // Form template field name: camelCase

form.capitalized.value = “TEST VALUE C” // Form template field name: Capitalized

form.uppercase.value = “TEST VALUE B” // Form template field name: UPPERCASE

form.lowercase.value = “TEST VALUE A” // Form template field name: lowercase
```

Also it is possible to represent Form attributed values as a Map. This allows easy access to the form data:

```
out << "List Form data as a Map <br>"

//List all form Records as a Map
submittedData.each {form ->
    out << "<br>"
    out << "${forms.formToMap(form)}"
}
```

Reverse logic is kept as well, meaning Form data cat be set from a Map utilizing `forms.MapToForm(Map map, Form form)`

Content Script: Create objects

Coming soon...¶

Integrate LLM services

Module Suite Training Center¶

What is it?¶

Module Suite Training Center is a simple Module Suite application that allows you to download and configure on your system a series of simple examples of using the Module Suite. The examples are organized into two main categories: Content Script and Beautiful Webforms and listed in increasing order of complexity.

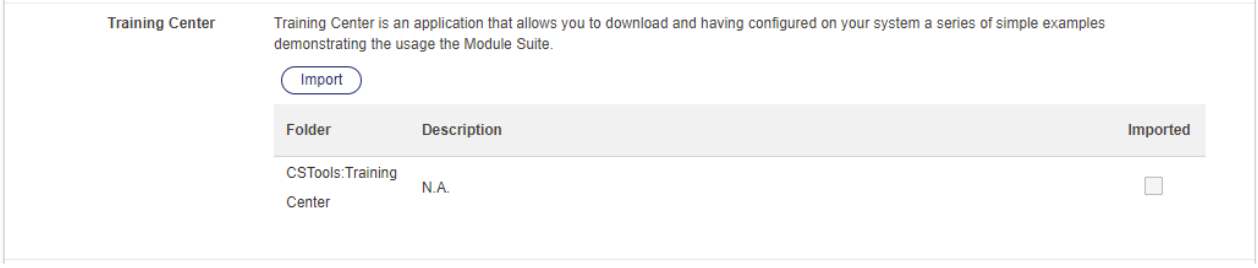
No Representations or Warranties; Limitations on Liability

The Training Center application (THE APPLICATION) has been created with the sole purpose of showcasing the Module Suite's capabilities. As such, it **should not be utilized in productive environments** and AnswerModules in no way guarantees that included examples are fully functional or free of errors. The information and materials on the Training Center application could include technical inaccuracies or typographical errors. Changes are periodically made to the information contained within it. AnswerModules Sagl MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO ANY INFORMATION, MATERIALS, CODES OR GRAPHICS ON THE APPLICATION, ALL OF WHICH IS PROVIDED ON A STRICTLY "AS IS" BASIS, WITHOUT WARRANTY OF ANY KIND AND HEREBY EXPRESSLY DISCLAIMS ALL WARRANTIES WITH REGARD TO ANY INFORMATION, MATERIALS CODES OR GRAPHICS ON THE APPLICATION, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT. UNDER NO CIRCUMSTANCES SHALL AnswerModules Sagl BE LIABLE UNDER ANY THEORY OF RECOVERY, AT LAW OR IN EQUITY, FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION, SPECIAL, DIRECT, INCIDENTAL, CONSEQUENTIAL OR PUNITIVE DAMAGES (INCLUDING, BUT NOT LIMITED TO LOSS OF USE OR LOST PROFITS), ARISING OUT OF OR IN ANY MANNER CONNECTED WITH THE USE OF INFORMATION OR SERVICE, OR THE FAILURE TO PROVIDE INFORMATION OR SERVICES, FROM THE APPLICATION.

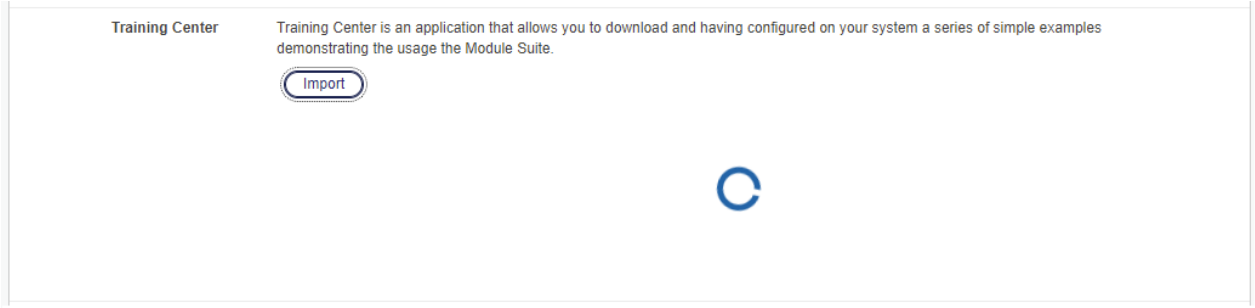
Training Center setup¶

Installing the Training Center application on your system is a straightforward procedure and can be performed using the Module Suite Content Script Volume Import Tool.

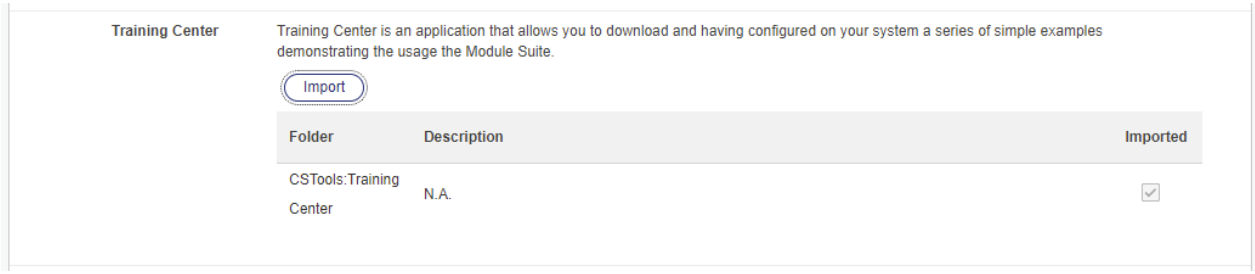
Within the Content Script Volume Import Tool, locate the section dedicated to CS Tools.



If the tool has not been installed yet (unchecked box on the right) proceed to install it by clicking on the "import" button.



Once complete, the Training Center tool will appear as "imported".



Using the tool¶

Internet access required

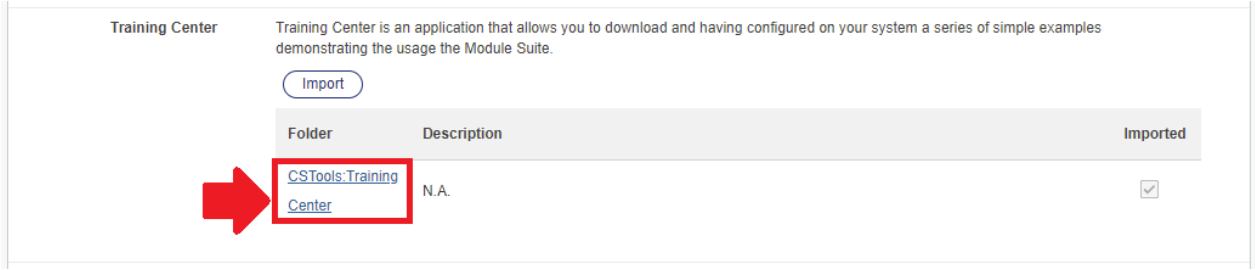
Your browser must have access to the Internet in order to properly execute the application of the Training Center.

As administrator

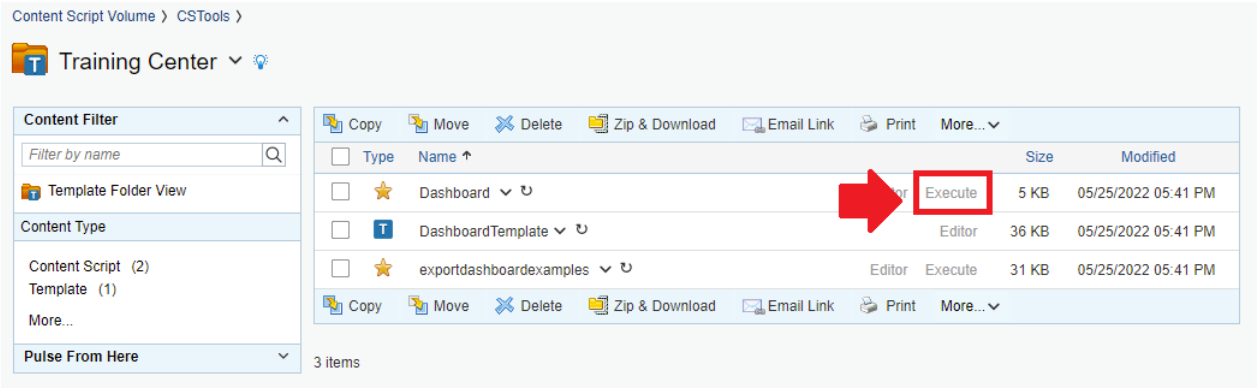
The examples must be imported using a user with administrative rights on the system (for example, the administrator user). Your browser is required to have access to the internet in order to be able to properly run the Training Center application.

In order to access the tool:

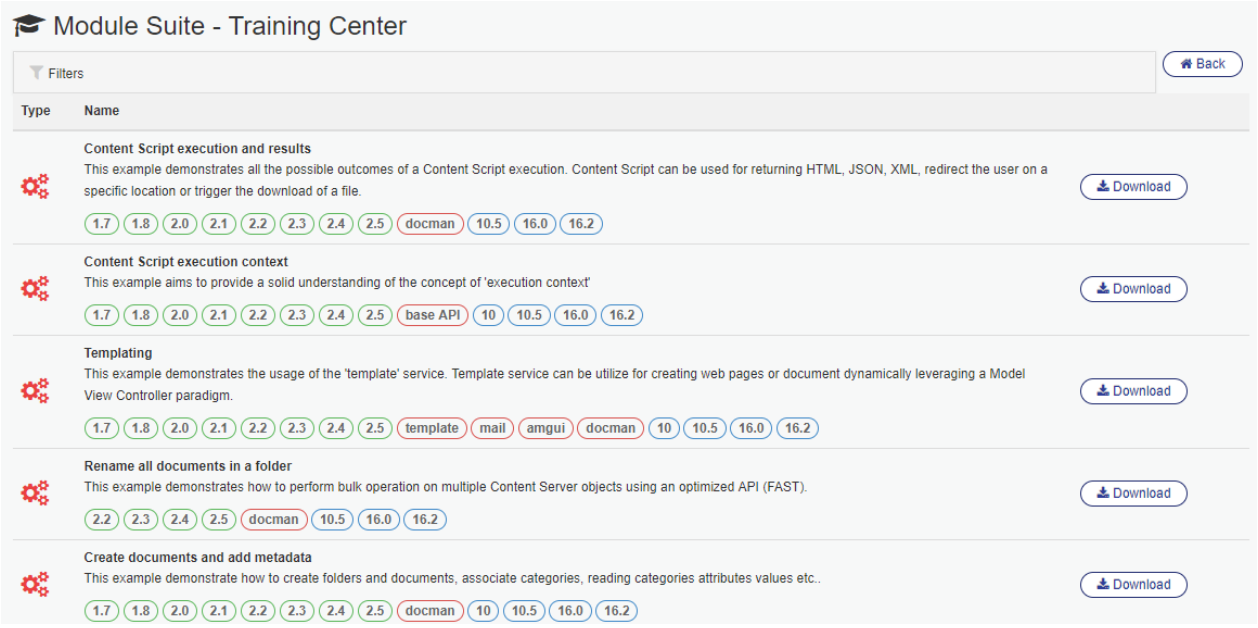
- navigate to the `Content Script Volume : CSTools : Training Center` folder.
- alternatively, from the Content Script Volume Import Tool, click on the **CSTools Training Center** link as shown below.



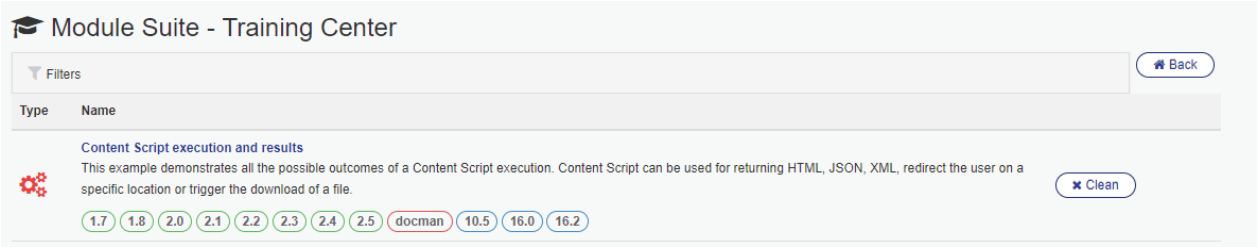
- execute the main **Dashboard** script to launch the tool.



To download and configure an example on your system just press the "download" button associated with it. The application will automatically download the required resources from the `developer.answermodules.com` portal and install / configure them on your system.



Once imported, the example will be available under `Enterprise:Module Suite examples`. Imported example are also directly accessible by clicking on the example title within the Training Center tool.



Do not manually delete imported examples

We strongly advise you not to manually delete any imported examples with the Training Center application. If you want to remove the example from your system, press the "clean" button associated with it (the application will perform the necessary cleanup steps on your behalf)

Tags¶

Following is a list of relevant tags:

CARL¶

- LLM services

Model Context Protocol¶

- MCP

OpenAI¶

- LLM services
- MCP
- OpenAI APIs

administration¶

- Administration tools
- Content Script Volume Import Tool

batch¶

- OpenAI APIs

clustered installation¶

- Installing on a clustered environment
- Upgrading a clustered environment

commands¶

- SmartPages commands

configuration¶

- Usage in Production

container¶

- Installing on containers

cost optimization¶

- OpenAI APIs

installation¶

- Apply Hotfixes
- Applying the license key manually
- Configure
- Content Script Volume Import Tool
- Deploying on Unix/Linux
- Deploying on Windows
- Getting Started
- Importing the license key
- Install
- Installing Beautiful WebForms
- Installing Content Script
- Installing Script Console
- Installing Smart Pages
- Installing on a clustered environment
- Installing on containers
- Upgrading a clustered environment

integration¶

- MCP

javascript¶

- SmartPages commands

llm¶

- LLM services
- OpenAI APIs

mcp¶

- MCP

performances-tips¶

- Content Script: Retrive information
- Usage in Production
- Widgets

productive¶

- Usage in Production

radio channel¶

- SmartPages commands

smartui¶

- SmartPages commands

uninstallation¶

- Uninstalling Module Suite

unix¶

- Deploying on Unix/Linux
- Installing Script Console

upgrade¶

- Content Script Volume Import Tool
- Getting Started
- Upgrading
- Upgrading a clustered environment