SCORM Engine 2013.1 Documentation

SCORM Engine Console: How It Should Look
- Import and Launch
- Tests
- Integration Details
- Web Server Configuration

How You Can Have a Console of Your Very Own
- Configuring Your Database for Use with SCORM Engine
- Configuring Your Web Server for Use with SCORM Engine
  - Controlling Access to Console
  - Getting SCORM Engine Talking to Your Database
  - Making the Web Server and the Filesystem Get Along
- Implementing the Integration Layer
  - When Worlds Collide: The SCORM Engine Override Functions

So You Want to Integrate SCORM Engine for .NET

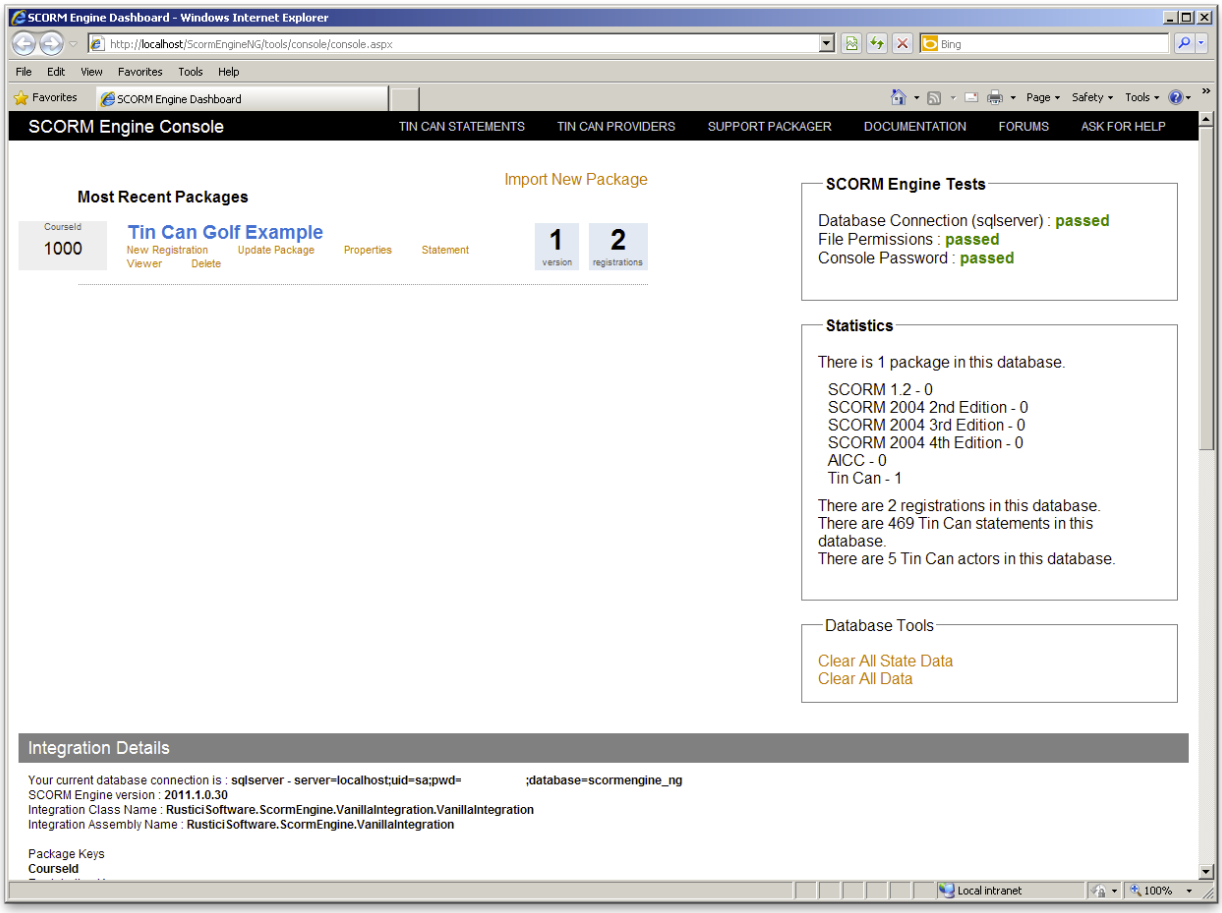
Welcome to SCORM Engine!
As a SCORM Engine customer, you'll have access to our development staff during the initial integration process and for as long as you maintain a support agreement. You'll also have access to tools that we hope will assist you for as long as you run SCORM Engine.
This document is geared toward what you'll see after the integration kickoff call, but the technologies covered will be available to you in your SCORM Engine installation for the duration of your use of the software.
As of the latest major release of SCORM Engine, we now offer a console to SCORM Engine. From the console, you can:

- import and launch content
- review registrations and launch history
- see a basic health check of your SCORM Engine environment
- get basic statistics about your use of supported learning standards
- get a snapshot of key integration details
- execute some basic database functions
- access a basic Tin Can statement viewer and manage OAuth consumers

By the time you're done reading this document and following its prescriptions, we want you to be able to use the console to import and launch content in a fully functional SCORM Engine integration. First, we take you through an overview of the SCORM Engine console. Then we tell you how to get yours up and running as you embark on the actual process of integration.
As you read and complete each step of this second portion of this document, you can track your own progress in the console dashboard. First you'll start seeing green lights in the self-test health check up. Ultimately, you'll be able to import and launch content.
Have fun!
(And be sure to let us know what could make this process better and more fun if you don't...)

SCORM Engine Console: How It Should Look

Pictured here is a screenshot of the SCORM Engine console dashboard running against a very basic integration for a sample customer called Vanilla:



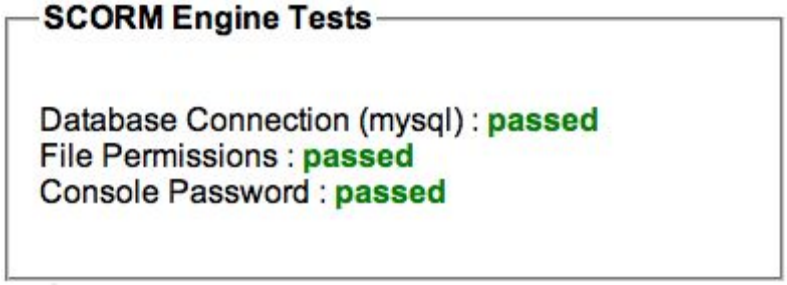
Import and Launch

You can see that we've imported into this instance of SCORM Engine one of our golf sample courses (which also shows as the only Tin Can course in the Statistics box over to the right).

Tests

You can also see in the SCORM Engine Tests box that we've got a valid database connection, that our courses directory is readable and writable, and that we've set a non-default password

for access to console.

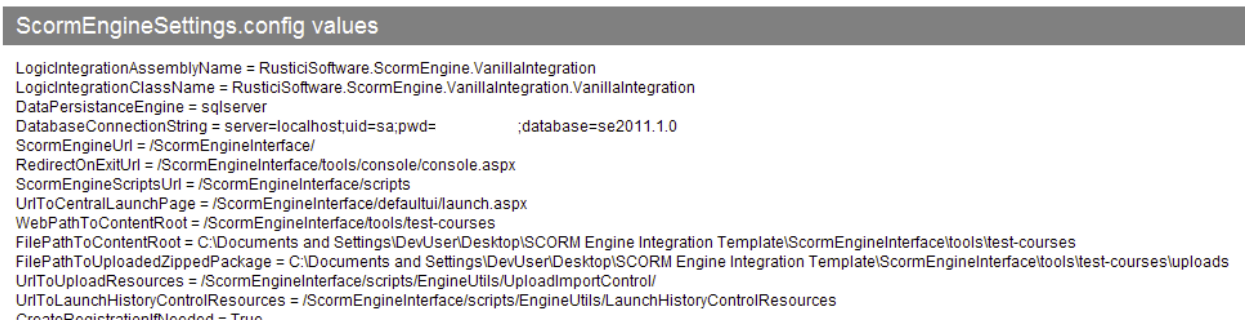


Integration Details

Under Integration Details at the bottom, you can see the database connection that is being used to drive this instance of SCORM Engine, the version of the SCORM Engine software, and the .NET files in use for the integration. You can also see that we've specified an external package key of CourseId, an external registration key of CourseId and Username, and no external configuration keys. Finally, you can see how your launch URLs will look.

Web Server Configuration

The final section on the console dashboard is for the web server configuration values relevant to successful operation of SCORM Engine. We display configuration settings as key/value pairs.



How You Can Have a Console of Your Very Own

In order to get to this point, we had to complete the basic integration process, which involved three steps:

running the SCORM Engine database script
making some basic changes to the SCORM Engine web server configuration file
implementing core overrides in the main integration file

.NET users have a slight advantage in that the comprehensive environment available in Visual Studio allows for console to be played directly in debugging mode right from Visual Studio rather than having to navigate to its URL via a web browser. If you want to access console directly, however, it's also available at /ScormEngineInterface/tools/console/console.aspx.

Configuring Your Database for Use with SCORM Engine

As part of our delivery, we include a SQL file containing data definition language (DDL) statements (e.g., CREATE TABLE, etc.) for the SCORM Engine data model.
You'll need to execute this collection of SQL statements in the DBMS (e.g., SQL Server, MySQL, Oracle, PostgreSQL) you're using with SCORM Engine. In our Vanilla example, we're using SQL Server. You'll find the SQL for your DBMS in the Database folder in your SCORM Engine folder. Each DBMS has its own subfolder.
You should only need to run this SQL once for each instance of SCORM Engine.

Configuring Your Web Server for Use with SCORM Engine

We include a web server configuration file specific to SCORM Engine in both our .NET (SCORMEngineSettings.config) and Java (SCORMEngineSettings.properties) releases. You'll need to update this configuration file to get both SCORM Engine and the console working.
In these examples, we'll be using the syntax for IIS, but the key/value pairs are the important pieces, and they're fairly consistent across our .NET and Java releases. And if you ever have any questions, just ask!
This first round of settings will be in the <appSettings> block of SCORMEngineSettings.config.

Controlling Access to Console

Console is your gateway to SCORM Engine. It includes both information and controls that you probably don't want the entire world to see, so we protect it with an authentication mechanism

that uses a combination of a configuration key and a cookie.
The very first time you try to access console, it will look like this:

That password is governed by the ConsolePassword entry in your config. In the appSettings block, you'll want an entry like this:

```
<add key="ConsolePassword" value="YourChosenPassword"/>
```

After you've authenticated successfully, console will set a cookie, and you will be able to bypass the login prompt and get straight to the dashboard.
If you ever want to disable access via a given browser that has previously authenticated, you'll need to delete the SECONSOLE cookie.

Getting SCORM Engine Talking to Your Database

In SCORMEngineSettings.config, we need to specify the connection details for the database set up in step 1. We'll need to adjust the entries for DataPersistenceEngine and DatabaseConnectionString. In our example, we use these values:

```
<!-- Data Persistence -->
<add key="DataPersistanceEngine" value="sqlserver"/>
<add key="DatabaseConnectionString"
value="server=localhost;uid=sa;pwd=notarealpassword;database=se2011.1.0"/>
```

DataPersistanceEngine just specifies the DBMS being used (i.e., sqlserver, mysql, oracle, or db2). The DatabaseConnectionString needs the basics for a database connection: a hostname, a user ID, a password, and the name of the database where SCORM Engine will live (and where you should've run the DDL statements referenced earlier in this document).
We are using SQL Server's `sa` user in our example. If you have a database user that you'll be using for SCORM Engine, just make sure it's in the connection string.
Once you've completed this step, you should be able to play the SCORM Engine Console in Visual Studio and see a green passed in the Database Connection test on the dashboard.

Making the Web Server and the Filesystem Get Along

In order to get import working, you'll need to make sure the web server has somewhere on the filesystem to put content files. There are four values you'll want to set in order for SCORM Engine to be able to import successfully:

WebPathToContentRoot — the URL (can be an absolute path without the

protocol or server) to web-accessible folder where your content will live
FilePathToContentRoot — the full filesystem file path to where the same content exists on the server
FilePathToUploadedZippedPackage — the full filesystem file path to where zipped packages will be uploaded before being unzipped, imported, and moved to the content root
UrlToUploadResources — the URL (can be an absolute path without the protocol or server) indicating where your upload mechanism lives

In the case of the two filepaths, these need to be writable by the web server.
Here's how this section of the appSettings block looks in SCORMEngineSettings.config for our Vanilla integration:

```
<!-- Upload Import Control -->
<add key="WebPathToContentRoot" value="/ScormEngineInterface/tools/test-courses"/>
<add key="FilePathToContentRoot" value="C:\Documents and Settings\DevUser\Desktop\SCORM Engine Integration Template\ScormEngineInterface\tools\test-courses"/>
<add key="FilePathToUploadedZippedPackage" value="C:\Documents and Settings\DevUser\Desktop\SCORM Engine Integration Template\ScormEngineInterface\tools\test-courses\uploads"/>
<add key="UrlToUploadResources" value="/ScormEngineInterface/scripts/EngineUtils/UploadImportControl/"/>
```

Once you've got these values set up in your configuraiton file with permissions to the directories such that the web server can write to them, you should have another passed test in the console dashboard.

Implementing the Integration Layer

As part of our delivery, we ship you four files that constitute your integration layer:

a file containing your external configuration, i.e., information your LMS might want to use in SCORM Engine that is available to all integration functions (VanillaExternalConfiguration.cs in our example)
a file containing your external package ID, the keys we'll use to uniquely identify your content during import and launch (VanillaExternalPackageId.cs in our example)
a file containing your external registration ID, the key(s) we'll use to uniquely match a learner to a launch (VanillaExternalRegistrationId.cs in our example)
a file containing your integration override functions (VanillaIntegration.cs in our example)

We should have delivered you versions of each of the external files with the keys already populated, but you will always be able to see their status in console.
And you'll have stubs of the core override functions in your main integration file, but you will need to complete implementation of these overrides in order to have a complete integration

between your LMS and SCORM Engine.

When Worlds Collide: The SCORM Engine Override Functions

The core override functions go in our main integration file, VanillaIntegration.cs:

GetLearnerInformation() — gets the learner name and ID from the host LMS for storage in SCORM Engine

AddExternalPackage() — required when you're computing the package keys in the host LMS rather than passing them directly via the upload/import control

RollupRegistration() — persists data to the SCORM Engine server at regular intervals (by default every 10 seconds)

RollupRegistrationOnExit() — persists data only upon completion of the content (e.g., return to LMS)

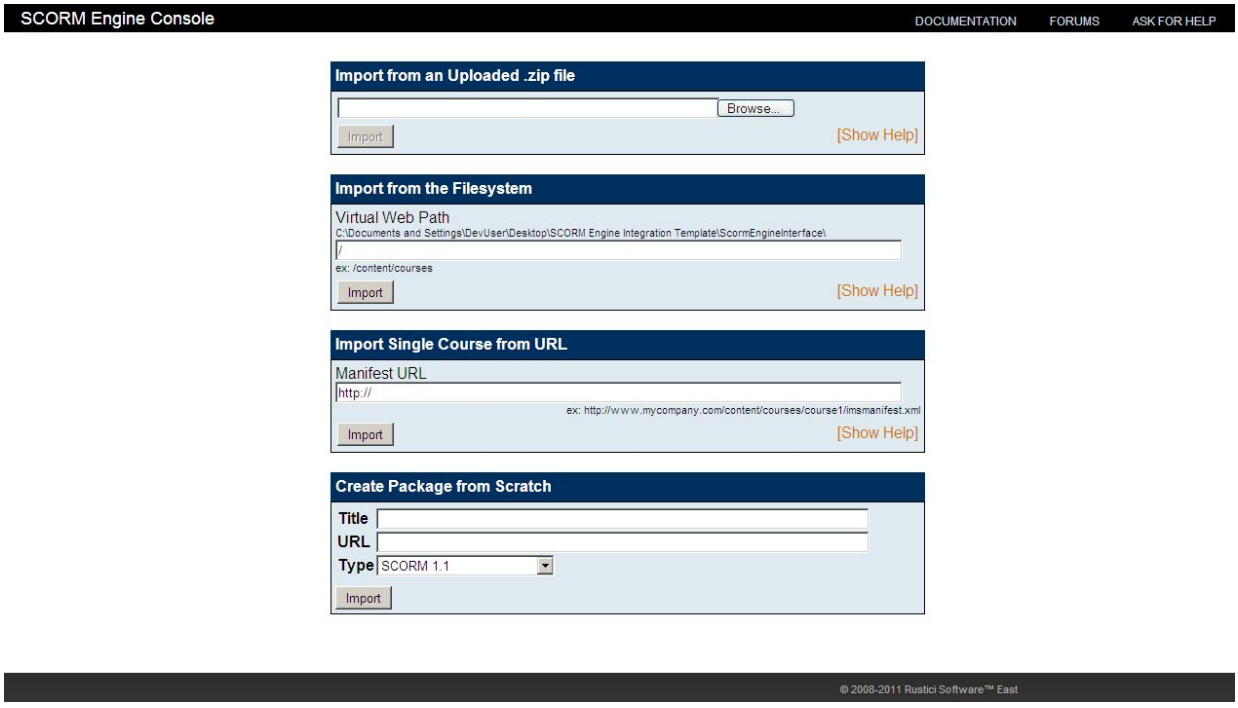
GetExternalPackageIdFromExternalRegId() — required to correctly identify content in integrations where the registration keys do not include the package keys

As mentioned above, we provide stubs for each of these functions upon delivery of your integration code (including only definitions for the ones that matter for your integration; you might not need GetExternalPackageIdFromExternalRegId(), for instance), but you'll likely need to customize them. For instance, if you're tracking learning in your LMS, you'll need the logic for storing that tracking information in your non-SCORM Engine LMS database to be included in your rollup override(s).

Your First Import

As a part of your SCORM Engine delivery, you'll find an example import file (import.aspx) in /ScormEngineInterface/tools/console. This is how console will allow you to test your imports, but you can also model your integrated import process on it. It uses an upload/import control we provide that is available for you to use to integrate imports however you like.

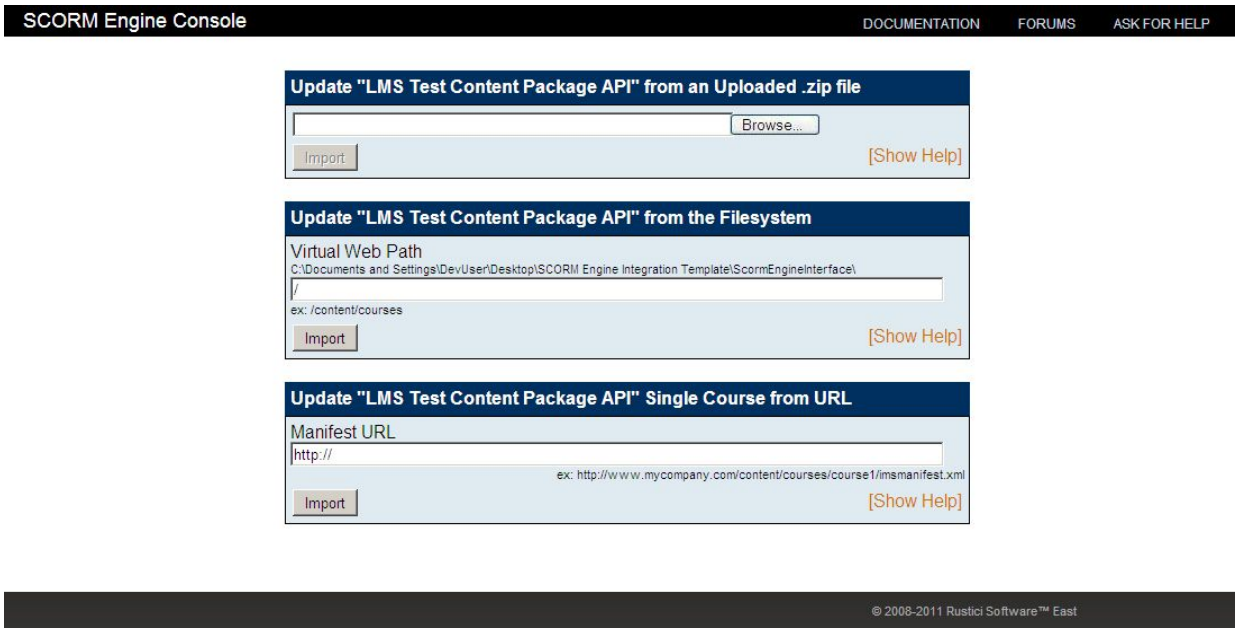
Here is how the import screen will look in console (and if you use our default upload/import control in your final integration):



When you import, you might already have your external package ID available. If so, you can just pass that in to the import process. In console, if you've got a package ID already, you add it to the query string on import.aspx. E.g.,
import.aspx?package=CourseId|123
Then reload import.aspx. After you reload, when you submit the import form it will be able to grab the package ID from the query string and complete the import.
If you don't have your external package ID at the time of import, and instead prefer that the import process generate one for you, we provide an overridable method called AddExternalPackage(). In this method, you can grab the title and description and whatever else you might need from the manifest and store them to your host system while also generating the package ID.
We provide you with sample code for this method. If you're not using it, we typically leave the code commented out and throw an exception to remind you that you need to pass in the package ID during the import process.

Updating Content

SCORM Engine comes with content versioning built in, so you can update a package in place without creating multiple parallel instances of your content. If you click Update Package under a course title in console dashboard, you'll see a screen like this:

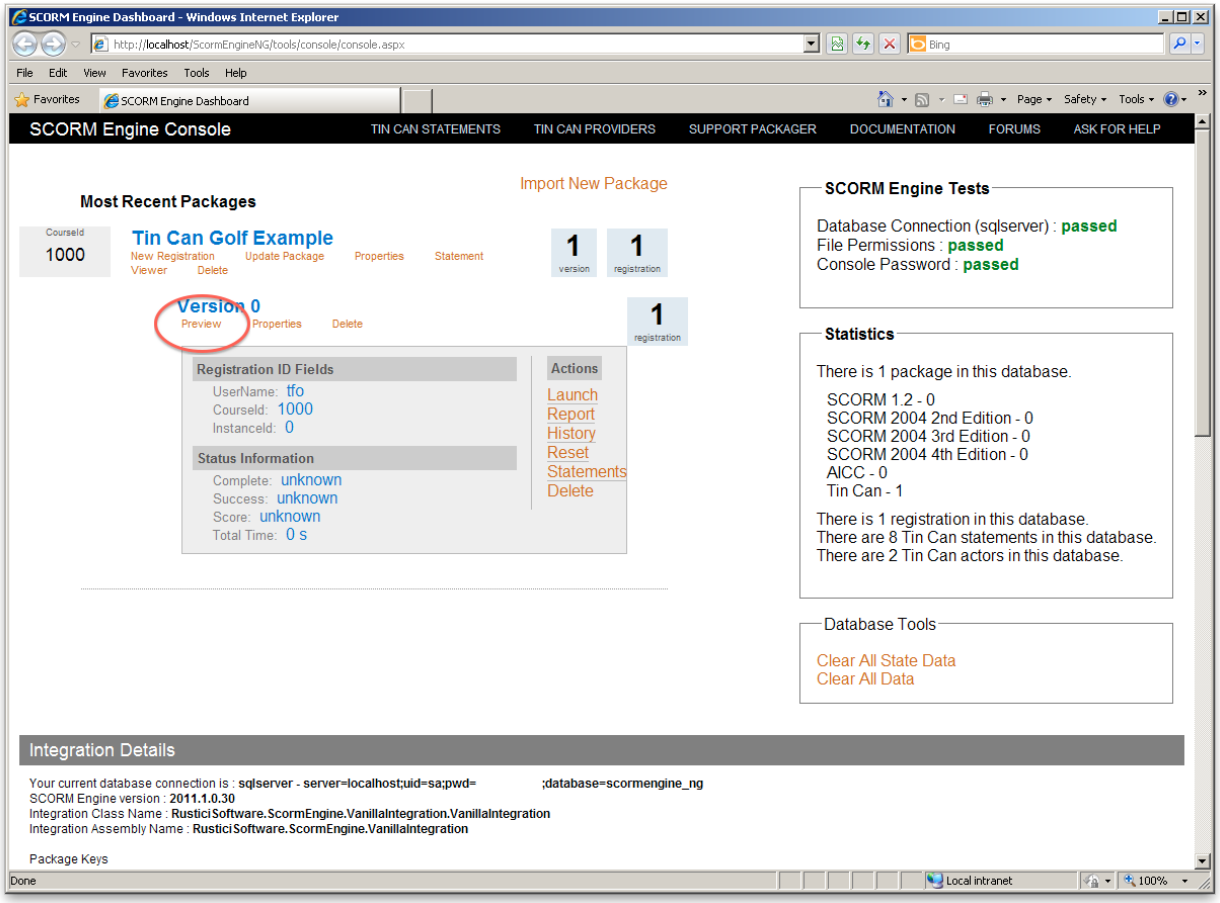


You'll notice there are only three options now. It doesn't really make sense to create a package from scratch if you're in the process of updating a pre-existing package.

Your First Launch: Preview

After you've successfully imported a course, even if you haven't completed your core overrides for tracking learning, you're ready to test a preview launch.

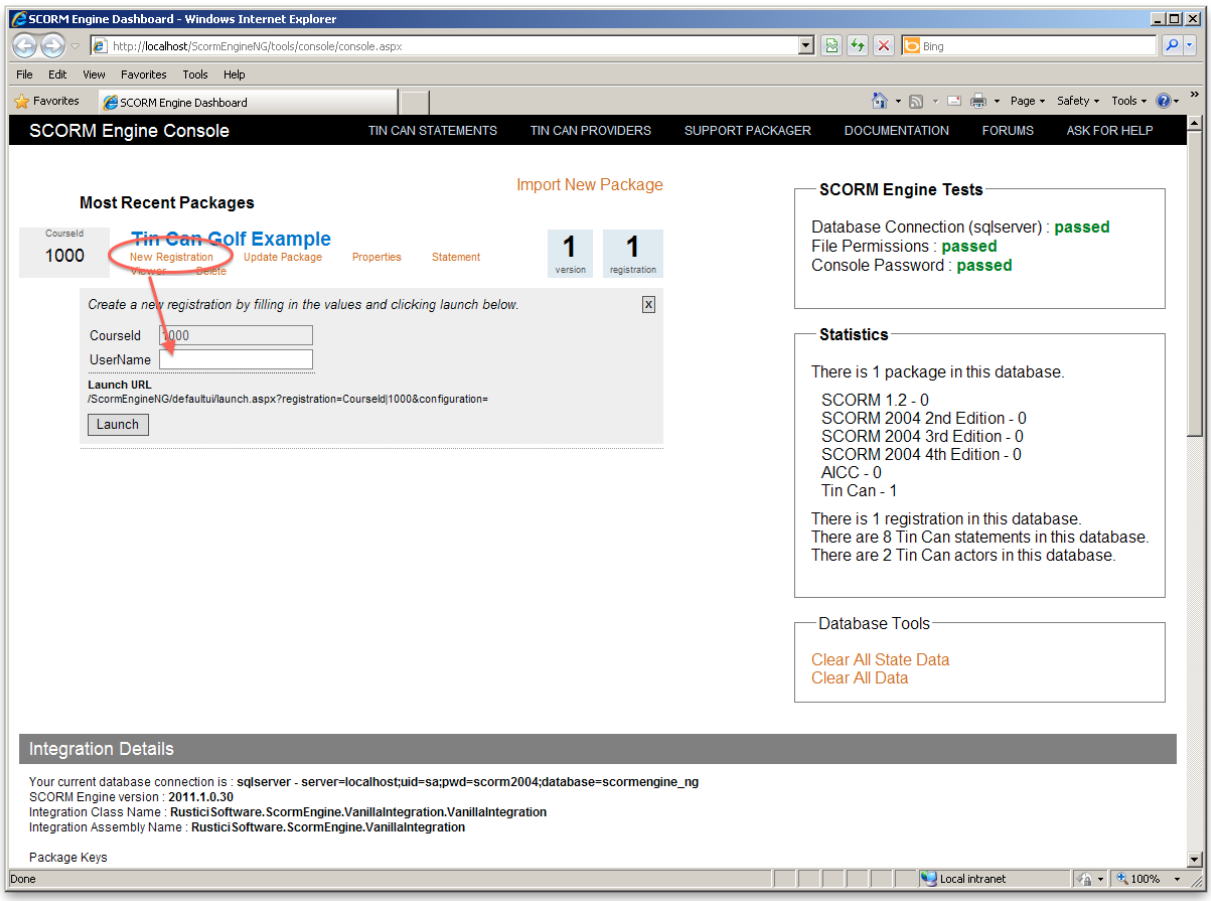
To test a preview launch, click on the title of an imported course. You'll see a link for "Preview." Click it, and you should be able to launch the course without worrying about learner information or rollup.



If you've gotten this far, you're in very good shape. It's sort of a metaphorical passed.

Your Second Launch: Tracking

After you've completed your GetLearnerInformation() and rollup override(s), you should be ready to test launching again, this time with SCORM Engine tracking learning.
To test a tracked launch, click the "New Registration" link beneath the title of one of your imported courses in the console dashboard. Now you should be able to track learning, which will include launch history and the ability to relaunch the content corresponding to this registration.



Beyond Console: Two Integrations Enter, One Integration Leaves

We provide SCORM Engine console to give you a snapshot of the functionality of your SCORM Engine setup and to serve as a sort of integration assistant. Getting it set up is almost like completing pre-integration.

At this point, all that likely remains for you to have a SCORM-conformant LMS is to integrate the upload/import controls as tightly as you'd like with your LMS and to ensure that your production environment is configured successfully (assuming that you set up SCORM Engine in a development environment to begin with).

If you started with console running in a dev environment, you'll still be able to run it in your production environment since it will run anywhere a successful ScormEngineInterface installation lives.

So You Want to Integrate

SCORM Engine for Java

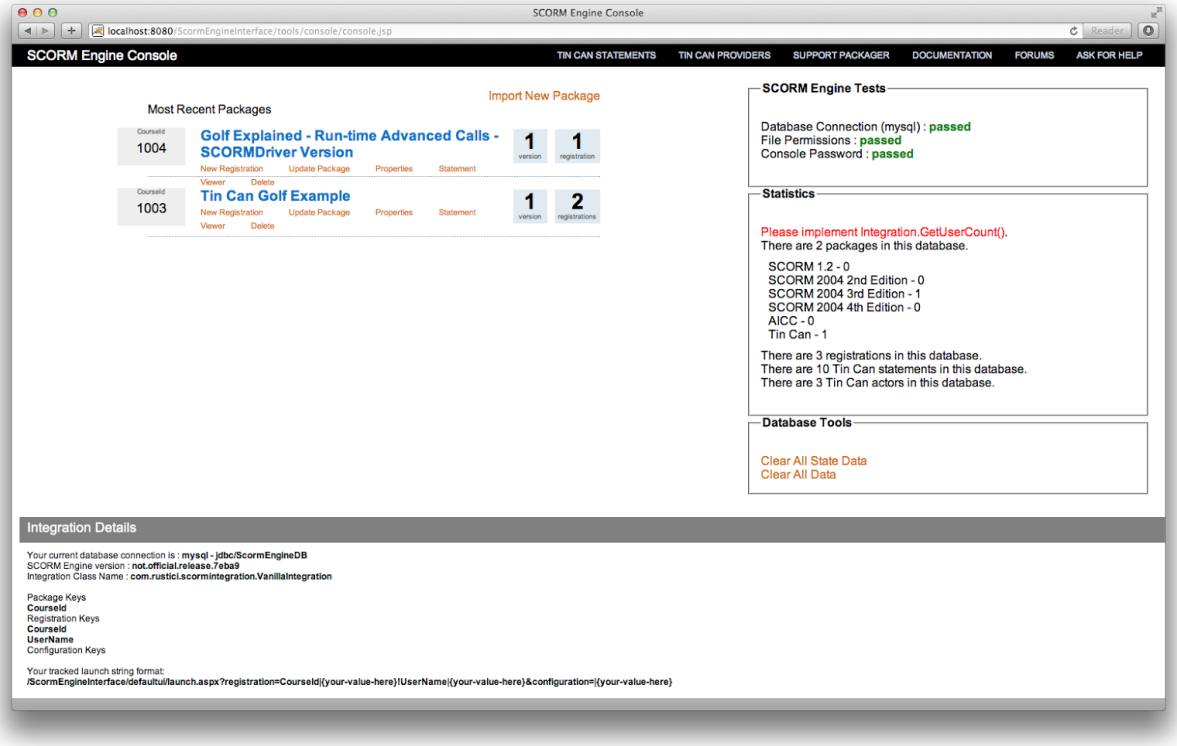
Welcome to SCORM Engine!
As a SCORM Engine customer, you'll have access to our development staff during the initial integration process and for as long as you maintain a support agreement. You'll also have access to tools that we hope will assist you for as long as you run SCORM Engine.
This document is geared toward what you'll see after the integration kickoff call, but the technologies covered will be available to you in your SCORM Engine installation for the duration of your use of the software.
As of the latest major release of SCORM Engine, we now offer a console to SCORM Engine. From the console, you can:

- import and launch content
- review registrations and launch history
- see a basic health check of your SCORM Engine environment
- get basic statistics about your use of supported learning standards
- get a snapshot of key integration details
- execute some basic database functions
- view Tin Can statements and manage OAuth consumers

By the time you're done reading this document and following its prescriptions, we want you to be able to use the console to import and launch content in a fully functional SCORM Engine integration. First, we take you through an overview of the SCORM Engine console. Then we tell you how to get yours up and running as you embark on the actual process of integration.
As you read and complete each step of this second portion of this document, you can track your own progress in the console dashboard. First you'll start seeing green lights in the self-test health checkup. Ultimately, you'll be able to import and launch content.
Have fun!
(And be sure to let us know what could make this process better and more fun if you don't...)

SCORM Engine Console: How It Should Look

Pictured here is a screenshot of the SCORM Engine console dashboard running against a very basic integration for a sample customer called Vanilla:

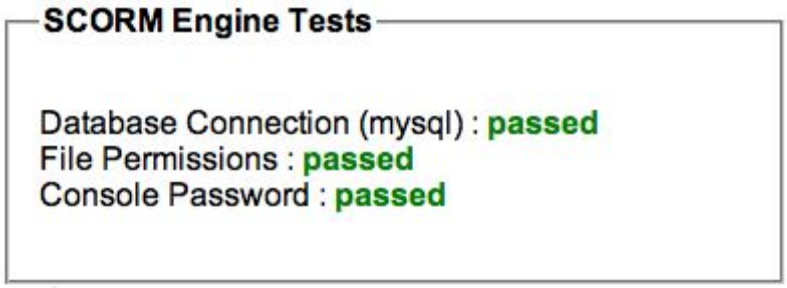


Import and Launch

You can see that we've imported into this instance of SCORM Engine a couple of our golf sample courses (which also show up in aggregate in the Statistics box over to the right).

Tests

You can also see in the SCORM Engine Tests box that we've got a valid database connection, that our courses directory is readable and writable, and that we've set a non-default password for access to console.

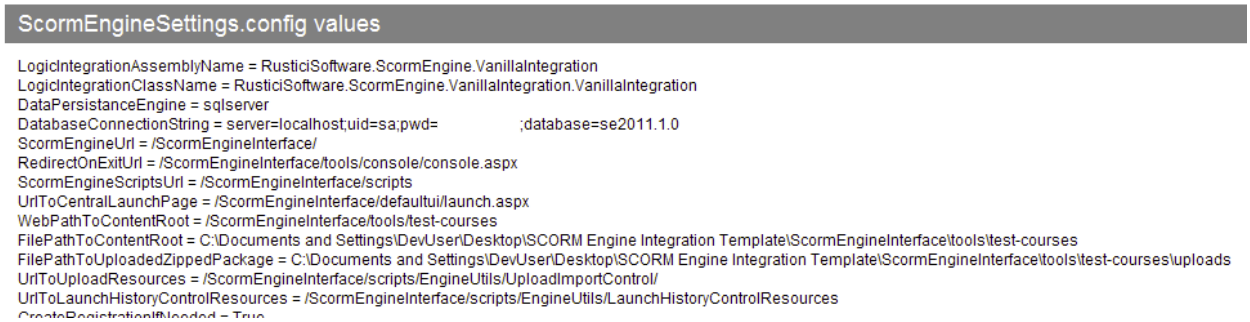


Integration Details

Under Integration Details at the bottom, you can see the database connection that is being used to drive this instance of SCORM Engine, the version of the SCORM Engine software, and the primary integration class in use. You can also see that we've specified an external package key of CourseId, an external registration key of CourseId and Username, and no external configuration keys. Finally, you can see how your launch URLs will look.

Web Server Configuration

The final section on the console dashboard is for the web server configuration values relevant to successful operation of SCORM Engine. We display configuration settings as key/value pairs.



How You Can Have a Console of Your Very Own

In order to get to this point, we had to complete the basic integration process, which involved three steps:

running the SCORM Engine database script
making some basic changes to the SCORM Engine web application properties file
implementing core overrides in the main integration file

Once the basic steps are done, you can access console via your SCORM Engine URL at </ScormEngineInterface/tools/console/console.jsp>.

Configuring Your Database for Use with SCORM Engine

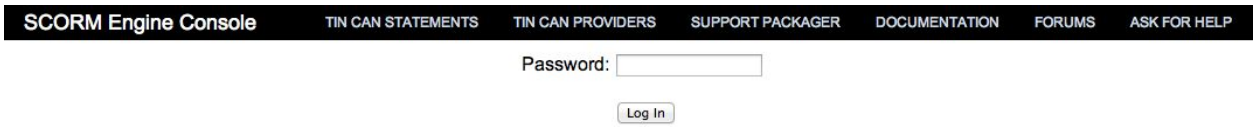
As part of our delivery, we include a SQL file containing data definition language (DDL) statements (e.g., CREATE TABLE, etc.) for the SCORM Engine data model.
You'll need to execute this collection of SQL statements in the DBMS (e.g., SQL Server, MySQL, Oracle, PostgreSQL) you're using with SCORM Engine. In our Vanilla example, we're using SQL Server. You'll find the SQL for your DBMS in the Database folder in your SCORM Engine folder. Each DBMS has its own subfolder.
You should only need to run this SQL once for each instance of SCORM Engine.

Configuring Your Web Server for Use with SCORM Engine

We include a web server configuration file specific to SCORM Engine in both our releases. You'll need to update this configuration file—SCORMEngineSettings.properties—to get both SCORM Engine and the console working.
In these examples, we assume you're using Tomcat as your application server, but if you ever have any questions, just ask!
All of these settings will be in the <properties> block of SCORMEngineSettings.properties.

Controlling Access to Console

Console is your gateway to SCORM Engine. It includes both information and controls that you probably don't want the entire world to see, so we protect it with an authentication mechanism that uses a combination of a configuration key and a cookie.
The very first time you try to access console, it will look like this:



That password is governed by the ConsolePassword entry in your properties file. In order to authenticate to console successfully, you'll need an entry like this:

```
<entry key="ConsolePassword">passw0rd</entry>
```

After you've authenticated successfully, console will set a cookie, and you will be able to bypass the login prompt and get straight to the dashboard.

If you ever want to disable access via a given browser that has previously authenticated, you'll need to delete the SECONSOLE cookie.

Getting SCORM Engine Talking to Your Database

In SCORMEngineSettings.properties, we need to adjust the entries for DataPersistanceEngine [yes, we recognize the typo] and DatabaseConnectionString. In our example, we use these values:

```
<!-- Data Persistence - DatabaseConnectionString *must* be a JNDI name for a DataSource-->
<entry key="DatabaseConnectionString">jdbc/ScormEngineDB</entry>
<!-- If you are connecting to an Oracle database then set DataPersistanceEngine to oracle,
     otherwise mysql should be fine (all other supported databases use the same jdbc interface)
-->
<entry key="DataPersistanceEngine">mysql</entry>
```

DataPersistanceEngine just specifies the DBMS being used (i.e., sqlserver, mysql, oracle, or db2). The DatabaseConnectionString needs to specify a JNDI name for a data source, which will actually be configured separately in Tomcat's conf/context.xml file, which we supply an example of in your integration deliverable.

Once you've completed this step, you should be able to play the SCORM Engine Console in Visual Studio and see a green passed in the Database Connection test on the dashboard.

Making the Web Server and the Filesystem Get Along

In order to get import working, you'll need to make sure the web server has somewhere on the filesystem to put content files. There are four values you'll want to set in order for SCORM Engine to be able to import successfully:

WebPathToContentRoot — the URL (can be an absolute path without the protocol or server) to web-accessible folder where your content will live

FilePathToContentRoot — the full filesystem file path to where the same content exists on the server

FilePathToUploadedZippedPackage — the full filesystem file path to where zipped packages will be uploaded before being unzipped, imported, and moved to the content root

UrlToUploadResources — the URL (can be an absolute path without the protocol or server) indicating where your upload mechanism lives

In the case of the two filepaths, these need to be writable by the web server.

Here's how this section of the appSettings block looks in SCORMEngineSettings.properties for our Vanilla integration:

```
<!-- Upload Import Control -->
<entry key="WebPathToContentRoot">/courses</entry>
<entry key="FilePathToContentRoot">/Library/WebServer/Documents/courses</entry>
<entry
key="FilePathToUploadedZippedPackage">/Library/WebServer/Documents/courses/uploads</e
ntry>
```

Once you've got these values set up in your configuraiton file with permissions to the directories such that the web server can write to them, you should have another passed test in the console dashboard.

Implementing the Integration Layer

As part of our delivery, we ship you four files that constitute your integration layer:

- a file containing your external configuration, i.e., information your LMS might want to use in SCORM Engine that is available to all integration functions (VanillaExternalConfiguration.java in our example)
- a file containing your external package ID, the keys we'll use to uniquely identify your content during import and launch (VanillaExternalPackageId.java in our example)
- a file containing your external registration ID, the key(s) we'll use to uniquely match a learner to a launch (VanillaExternalRegistrationId.java in our example)
- a file containing your integration override functions (VanillaIntegration.java in our example)

We should have delivered you versions of each of the external files with the keys already populated, but you will always be able to see their status in console.

And you'll have stubs of the core override functions in your main integration file, but you will need to complete implementation of these overrides in order to have a complete integration between your LMS and SCORM Engine.

When Worlds Collide: The SCORM Engine Override Functions

The core override functions go in our main integration file, VanillaIntegration.java:

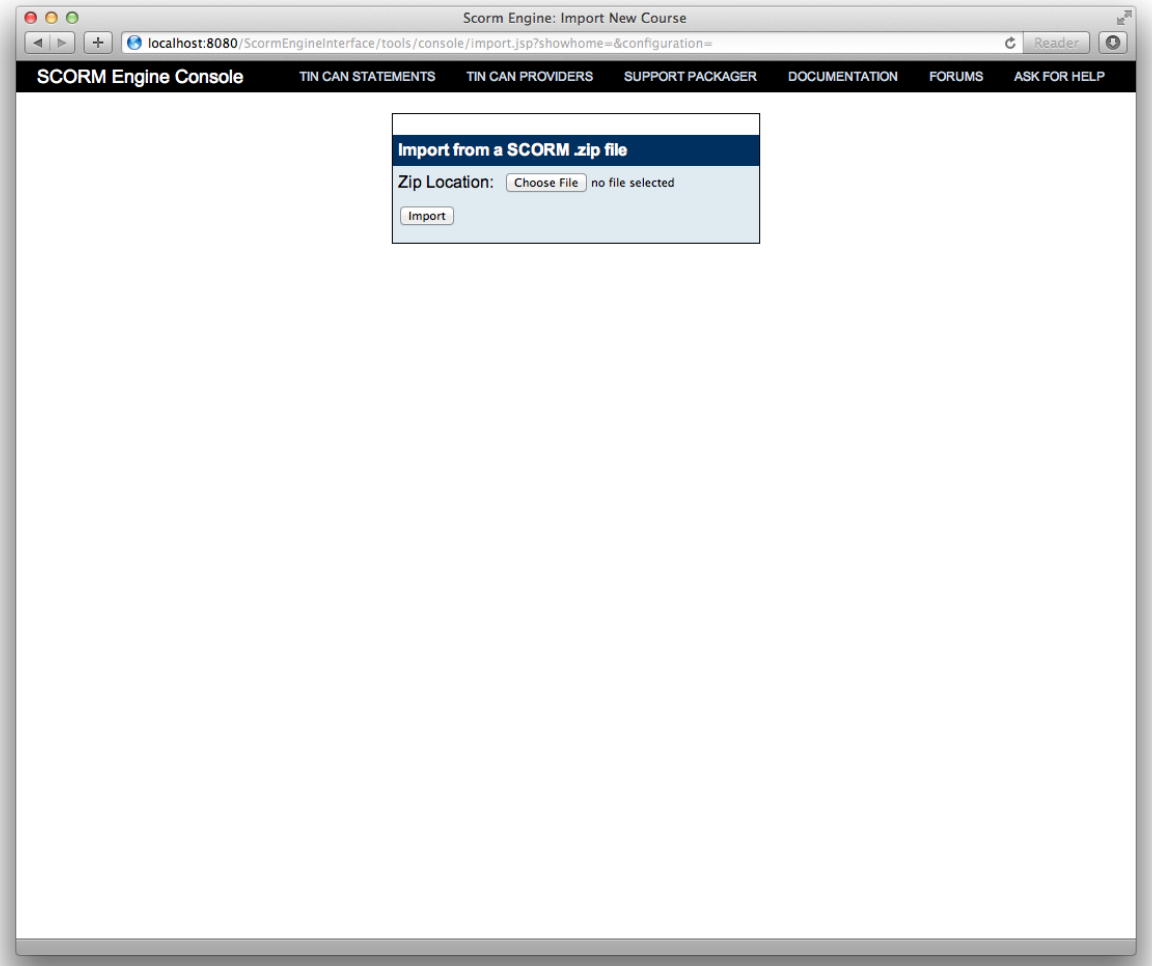
- **GetLearnerInformation()** — gets the learner name and ID from the host LMS for storage in SCORM Engine
- **AddExternalPackage()** — required when you're computing the package keys in the host LMS rather than passing them directly via import

RollupRegistration() — persists data to the SCORM Engine server at regular intervals (by default every 10 seconds)
RollupRegistrationOnExit() — persists data only upon completion of the content (e.g., return to LMS)
GetExternalPackageIdFromExternalRegId() — required to correctly identify content in integrations where the registration keys do not include the package keys

As mentioned above, we provide stubs for each of these functions upon delivery of your integration code (including only definitions for the ones that matter for your integration; you might not need GetExternalPackageIdFromExternalRegId(), for instance), but you'll likely need to customize them. For instance, if you're tracking learning in your LMS, you'll need the logic for storing that tracking information in your non-SCORM Engine LMS database to be included in your rollup override(s).

Your First Import

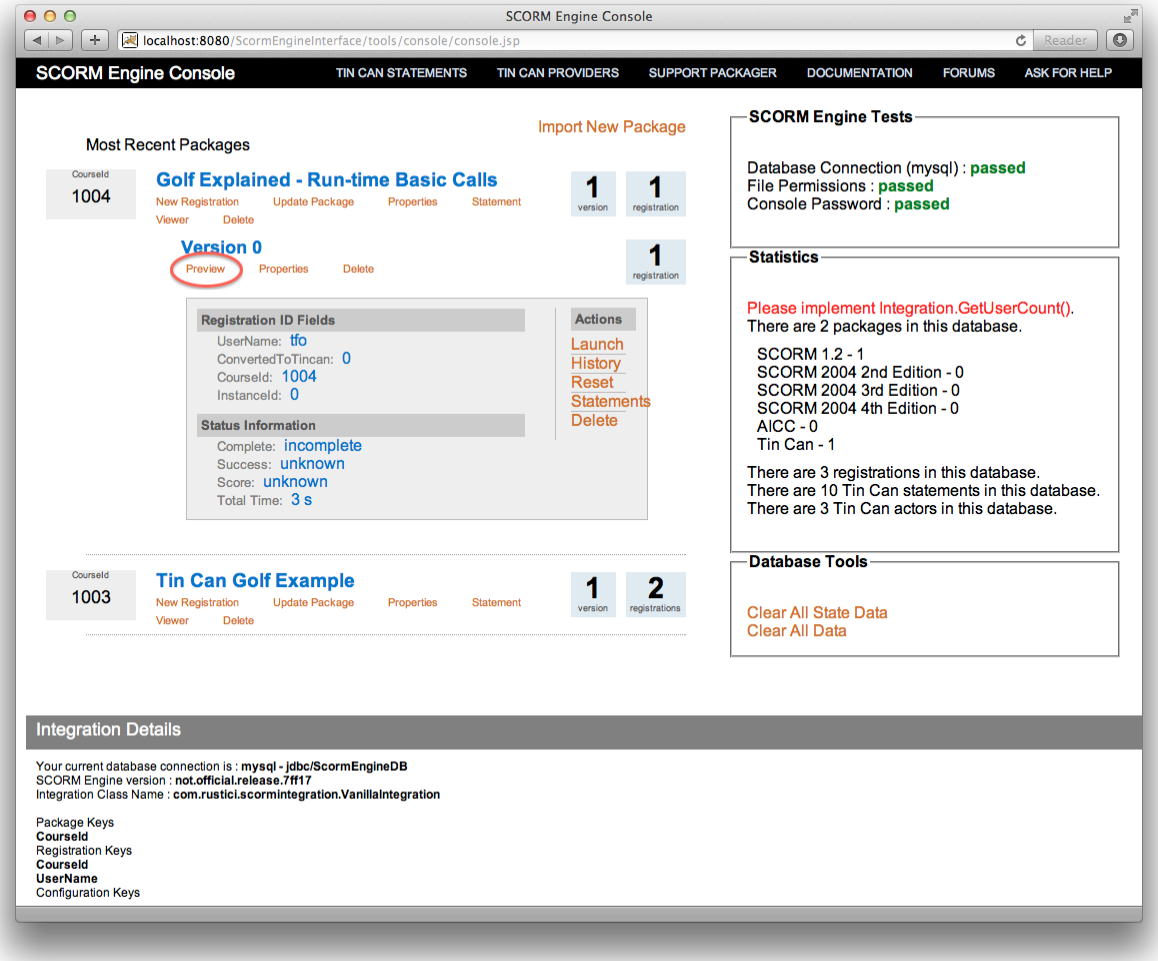
As a part of your SCORM Engine delivery, you'll find an example import file (import.jsp) in /ScormEngineInterface/tools/console. This is how console will allow you to test your imports, but you can also model your integrated import process on it. It uses an upload/import form we provide that is available for you to use as a sample to integrate imports however you like.
Here is how the import screen will look in console (and if you use our default import form in your final integration):



When you import, you might already have your external package ID available. If so, you can just pass that in to the import process. In console, if you've got a package ID already, you add it to the query string on import.jsp. E.g.,
import.jsp?package=CourseId|123
Then reload import.jsp. After you reload, when you submit the import form it will be able to grab the package ID from the query string and complete the import.
If you don't have your external package ID at the time of import, and instead prefer that the import process generate one for you, we provide an overridable method called AddExternalPackage(). In this method, you can grab the title and description and whatever else you might need from the manifest and store them to your host system while also generating the package ID.
We provide you with sample code for this method. If you're not using it, we typically leave the code commented out and throw an exception to remind you that you need to pass in the package ID during the import process.
SCORM Engine comes with content versioning built in, so you can update a package in place without creating multiple parallel instances of your content.

Your First Launch: Preview

After you've successfully imported a course, even if you haven't completed your core overrides for tracking learning, you're ready to test a preview launch.
To test a preview launch, click on the title of an imported course. You'll see a link for "Preview." Click it, and you should be able to launch the course without worrying about learner information or rollup.

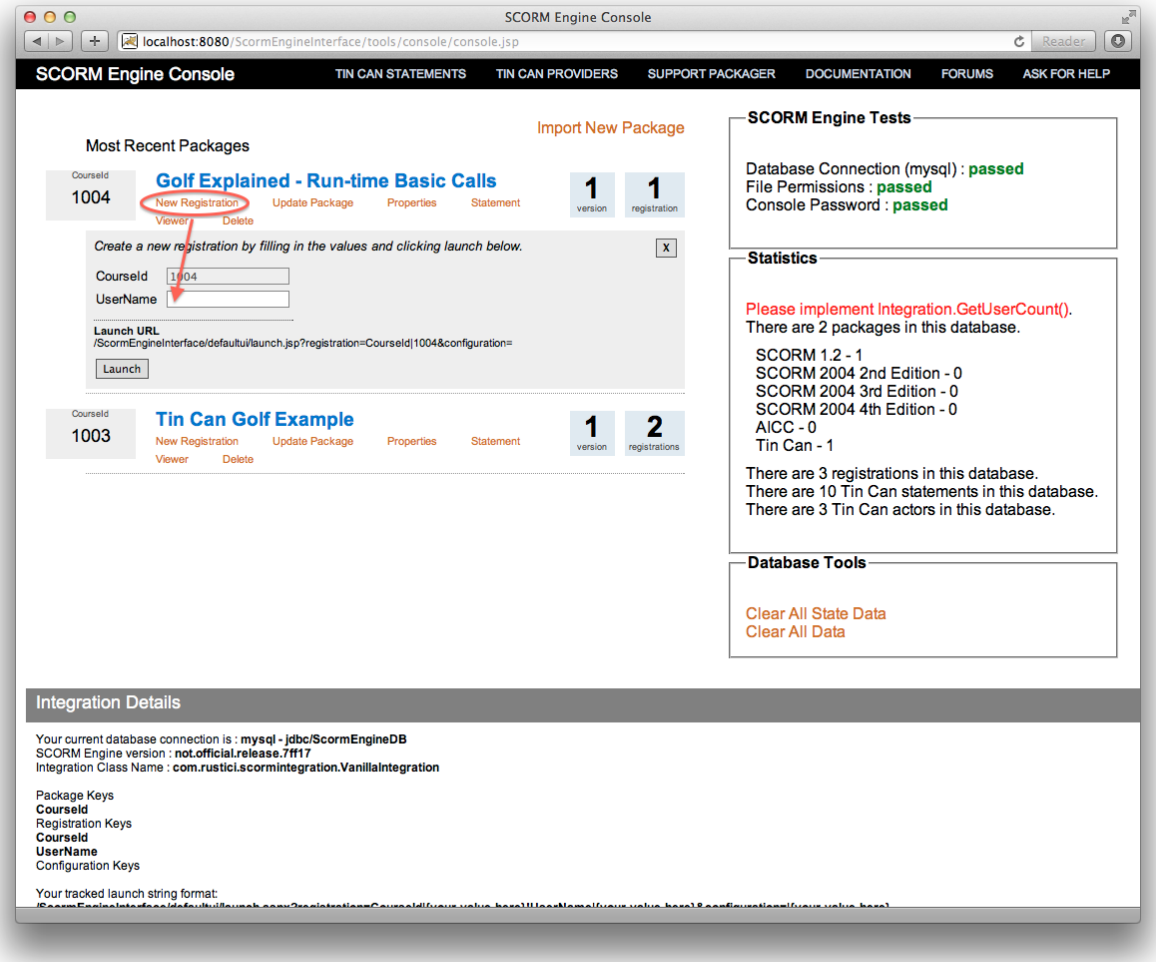


If you've gotten this far, you're in very good shape. It's sort of a metaphorical passed.

Your Second Launch: Tracking

After you've completed your GetLearnerInformation() and rollup override(s), you should be ready to test launching again, this time with SCORM Engine tracking learning.

To test a tracked launch, click the “New Registration” link beneath the title of one of your imported courses in the console dashboard. Now you should be able to track learning, which will include launch history and the ability to relaunch the content corresponding to this registration.



Beyond Console: Two Integrations Enter, One Integration Leaves

We provide SCORM Engine console to give you a snapshot of the functionality of your SCORM Engine setup and to serve as a sort of integration assistant. Getting it set up is almost like completing pre-integration.

At this point, all that likely remains for you to have a SCORM-conformant LMS is to integrate the upload/import form as tightly as you’d like with your LMS and to ensure that your production environment is configured successfully (assuming that you set up SCORM Engine in a development environment to begin with).

If you started with console running in a dev environment, you’ll still be able to run it in your production environment since it will run anywhere a successful ScormEngineInterface

installation lives.

Getting started with Tin Can API and SCORM Engine

What exactly is available with the new SCORM Engine?

Learning Record Store (LRS) capabilities with a fully functioning Tin Can API (TCAPI) endpoint, as well as a new web service that implements TCAPI
mapping of Tin Can activities to traditional SCORM Engine registrations, which allows TCAPI activities to launch and report to the SCORM Engine in a similar fashion as SCORM and AICC, thereby "playing nicely" in a traditional LMS workflow
automatic and on-demand generation of Tin Can statements from SCORM and AICC registrations
integrated Tin Can statement viewer

Upgrading from SCORM Engine 2012.2 (.NET)

Upgrading the SCORM Engine is trivial. Once done, here is how you can configure it for Tin Can and start playing with some real data.

Database Upgrade Script

As with most SCORM Engine upgrades, you'll need to update the db schema by running a script. Run `SCP_2012.2_UPGRADE_FROM_SCP_2012.1_SQLSERVER_SCP.sql` (or ORACLE, MYSQL, etc) in the tool of your choice.

Installing and Configuring SCORM Engine 2013.1

Basic installation for SCORM Engine 2012.2 doesn’t differ dramatically from previous versions. We’ve already discussed some of the new requirements created by our support for TCAPI. What follows are the basic steps to get SCORM Engine up and running out of the box.

Getting the SCORM Engine Files Set Up Correctly

At some point, you'll receive an official release of SCORM Engine 2012.x from us. This will typically look something like this: `scormengine_net_scorm2004_2012.2.0.2.zip` You'll want to unzip this archive to wherever your SCORM Engine web application will live. The ScormEngineInterface directory in our release archive is the one that will become your IIS web application. So you'll want to make sure that the ASP.NET user has full control of that directory under Properties->Security.

Creating Your SCORM Engine IIS Web Application

In the Internet Information Services app in Windows, you'll want to create a new virtual directory for ScormEngineInterface which we commonly name simply "scormengine".

Letting Your DBMS (e.g., SQL Server) Know about SCORM Engine

In order to run SCORM Engine, we have to execute some data definition language (DDL) statements in SQL to make sure that the tables our application needs are in your database. To do this, launch Microsoft SQL Server Management Studio (MSSMS). In the SQLSERVER directory, open the files `1-2012.2_SQLSERVER_SCORMENGINE.sql` and `2-2012.2_SQLSERVER_VANILLAINTEGRATION.sql`. If you already know which database your SCORM Engine tables and data will live, select it. Then execute the two files in sequence. If you're using one of the other DBMSes supported by SCORM Engine (MySQL, Oracle, or PostgreSQL), use the relevant tool and execute the same SQL in the corresponding directory for your DBMS.

Teaching IIS to Speak SCORM Engine

Even though you've already set up your SCORM Engine IIS web application, you still need to update the configuration that will be used by IIS so that it can play nicely with SCORM Engine. At a minimum, you'll need to update the following values in SCORMEngineSettings.config to reflect your local environment.

Upload/Import

WebPathToContentRoot: a URL that points to the top-level directory where your content will live after it's been uploaded and unzipped
FilePathToContentRoot: an absolute filepath to the same directory on your filesystem
FilePathToUploadedZippedPackage: an absolute filepath to the directory on your filesystem where your content will get uploaded before being unzipped

Database Connectivity

DataPersistanceEngine: the DBMS you're using (e.g., "sqlserver")
DatabaseConnectionString: the connection string required by that DBMS to create a valid connection

Console

ConsolePassword: console uses this value for basic authentication. After the first authentication, subsequent authorization is governed by a cookie.

Testing Your SCORM Engine Installation

Once you've gotten this far, you should be able to navigate to the SCORM Engine console directly in your ScormEngineInterface. E.g., something like this:
`[yoursite]/ScormEngineInterface/tools/console/console.aspx`
Provided you've got your database connection configured correctly, you should be prompted to log in. After you've used your `ConsolePassword` to log in, you should see the dashboard of your console. If both the database and filesystem tests are passing, you're ready to try importing content!
Try importing a Tin Can package. If the import works, return to the console dashboard and try launching it in preview mode. If preview mode works, try launching it with an actual registration. If that works, return to the console dashboard and see if you see corresponding statements in your Statement Viewer. If you do, then you're ready to be driven by SCORM Engine! Now it's time to work on your integration...

Security and the Tin Can API

Each statement that comes into the Tin Can web service is evaluated for access rights before proceeding. The first thing that's determined is the "Asserter". The Asserter is essentially a combination of an Actor and a set of permissions. The Actor here is the person/system that is

acting as the authority for the Tin Can statement being processed. When statements are being written, this Actor actually shows up as the authoritative source in the statement.
Tin Can security is fully customizable through new SCORM Engine Integration methods. If using basic authentication you will likely want to implement:

```
Actor TinCanGetAuthorityFromBasicAuth(TCAPIContext context, String username, String password);
```

The default implementation will only accept one username/password which has full authority. This name/password is defined by your SCORMEngineSettings.config entry named "TinCanRootAccount". This config entry has both the name and password separated by a colon. Ex: "joeadmin:mypass".
If using OAuth we already have a good default implementation so you probably won't override this, at least initially..
What a particular user can do is defined by the Integration method `TinCanGetPermissions()`. We have defaults for the root user, a person(actor) and an application(actor). However, by overriding this integration method you can have fine-grained control to all permissions.

SCORM Engine Integration Architecture

Last Modified: Jun 23, 2011

Current SCORM Engine Version: 2013.1

Background

The SCORM Engine integrates with many, many systems, over 80 different LMS's at current count. It provides a piece of vital functionality to these systems and **needs to be tightly integrated** to provide both a seamless user experience and a robust technical implementation.

While a tight integration is vital to the long term success of an LMS powered by the SCORM Engine, it is also important that these systems not be so interwoven that they cannot be maintained separately. Much of the value that Rustici Software provides its clients stems from our ability to maintain and improve the SCORM Engine as the standards evolve and new interpretations of them emerge. Similarly our clients must be able to evolve and improve their LMS's without being encumbered by the SCORM Engine or relying on Rustici Software to make code changes. Thus the SCORM Engine and the host LMS need to be remain logically separate systems. From a technical perspective, we say that the SCORM Engine **needs to be loosely coupled** with the host LMS.

While many LMS's are quite similar in their basic structure and concepts, each has its own set of functionality that makes it unique. Different sets of business rules, innovative features and even subtle quirks can all affect how SCORM content should best be delivered in a particular LMS. Thus, the SCORM Engine **needs to be highly customizable and configurable** to handle whatever is thrown at it.

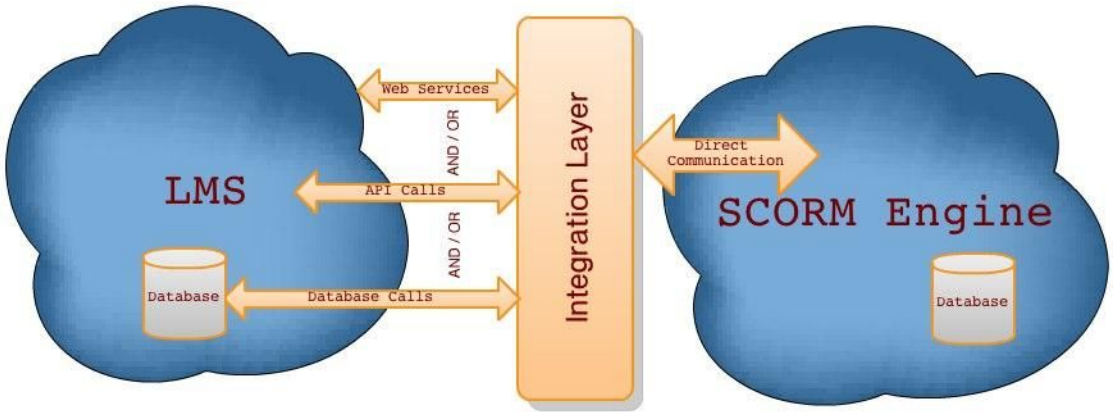
So, the SCORM Engine needs to be tightly integrated, loosely coupled and highly customizable. Those three goals are often at odds with one another. It took some

innovative design work to craft an architecture for the SCORM Engine that meets these requirements, but the effort has paid off. The "integration layer" architecture described in this document has enabled us to integrate the SCORM Engine with dozens of different LMS systems without ever having to make a major change to the core SCORM Engine code.

The Integration Layer

The integration layer is the interface between the SCORM Engine and the system with which it is integrating (the "host LMS"). It is also the boundary between the two systems, acting as a buffer to keep the core systems separate.

(This document will refer to the "system with which the SCORM Engine is integrating" as the "host LMS". We use this term for convenience, but note that while most integrations are with an LMS, the SCORM Engine has been integrated with a number of systems not directly related to learning.)



Loosely Coupled

The diagram above depicts the conceptual architecture of the SCORM Engine integrated with an LMS through the integration layer. Notice that the SCORM Engine does not directly communicate with the host LMS. Instead, all communication is routed through the integration layer. The common interface of the integration layer provides a level of indirection that isolates the host LMS from changes in the SCORM Engine and vice versa.

The integration layer is different for every integration of the SCORM Engine. The integration layer is also the *only* thing that is different for each integration of the SCORM Engine. The integration layer can be thought of as the "stuff we change" or the "stuff you are allowed to touch" when integrating.

Tightly Integrated

Notice that there is a very tight integration between the SCORM Engine and the integration layer. The integration layer is essentially a component of the SCORM Engine that can be swapped out for each integration. The interface between the integration layer and the host LMS can be very tight or very loose. The integration layer can communicate very loosely with the host LMS via web services (or even URL redirections). Or, the integration layer can invoke an LMS-provided API for a tighter integration. The integration layer can even make direct calls into the host LMS's database to achieve an extremely

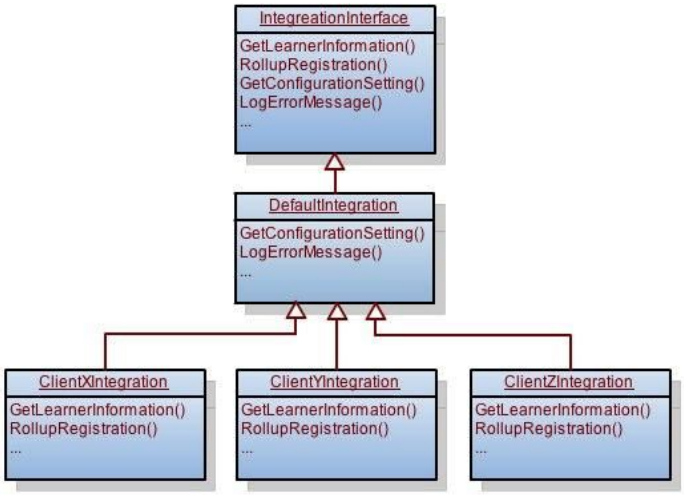
tight integration. All of these solutions are perfectly viable and it is up to the client to decide how tight a particular integration should be.

Highly Customizable

The integration layer is also where we can customize and configure the SCORM Engine. For anything that ever has been, or conceivably could be, customized in the SCORM Engine, there is an integration function that lets the SCORM Engine "ask" the integration how the action should be performed. For instance, before displaying the user interface, the SCORM Engine will ask the integration layer "which skin should I show?". Or, before writing a log message, the SCORM Engine will ask "where should I write this message to?". There are well over 100 integration functions that let us precisely customize the SCORM Engine for a particular host LMS.

How It Works

The core of the integration layer is an abstract class called the integration interface. The integration interface defines all the operations the SCORM Engine needs to perform which might vary based on the particular integration. For each integration, we create a unique class that implements all the methods defined in the integration interface. The method implementations in this subclass are specific to the host LMS and implement the functionality the client needs. At runtime, the SCORM Engine uses a factory class to instantiate the appropriate integration implementation.



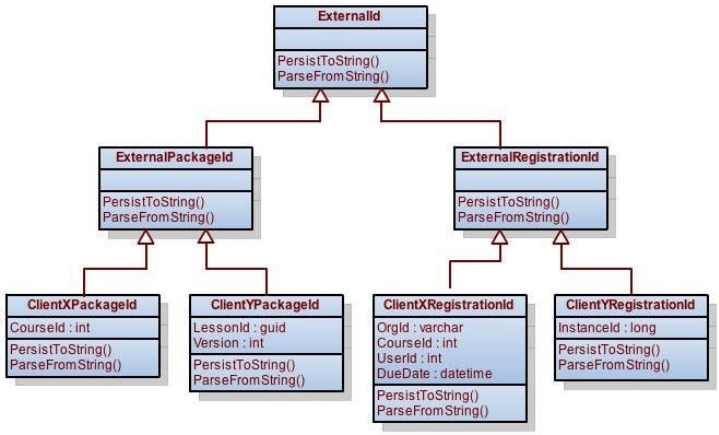
The UML diagram above depicts the class hierarchy for the integration classes. The boxes represent classes and the arrow indicate inheritance. At the top, is the abstract IntegrationInterface class. This class is where all of the integration methods are defined (but not implemented). At the bottom, there is are many concrete implementations of the the IntegrationInterface. Each client has their own unique implementation with all of the required methods implemented.

In the middle, there is a DefaultIntegration class. The purpose of this intermediate class is to provide default implementations of the many methods in the IntegrationInterface that usually do not change from client to client. There are over 100 integration methods defined in the IntegrationInterface. Of these, only about a dozen are *required* to change from client to client. The rest of the methods are there to *allow* things to change between

clients, but in most cases there is a default implementation that is perfectly acceptable. For example, in most cases, it is perfectly acceptable to log error messages to the standard event log. On the other hand, some LMS's have their own built in event tracking system, in which case it would be appropriate to override the default logging mechanism.

Data Relations

One of the core principles of the SCORM Engine integration design is that the host LMS should not need to know anything of the internals of the SCORM Engine. This separation helps to maintain the loose coupling between systems. Yet, many of the integration functions require that the systems communicate about a specific package or a specific registration. Rather than requiring the host LMS to know of the SCORM Engine's internal identifiers, the SCORM Engine defines a set of external identifier classes that allow the host LMS to use its existing identifiers no matter their structure and type.



Every LMS uses a different set of identifiers (each with different data types) to represent packages and registrations. Some use integers, some use strings, some use GUIDs, some use a combination of all of these and more). Some LMS's refer to packages as courses, others as lessons or tasks or items or classes. The integration layer defines an abstract way to represent these complex and varying objects in a consistent manner through the ExternalPackageId and ExternalRegistrationId classes.

When we create an integration we generate a concrete implementation of the ExternalPackageId and ExternalRegistrationId that is unique to the host LMS. These integration objects will have a set of properties that mirrors the keys used by the host LMS for the identified package and registration entities. These objects give the host LMS the ability to communicate with the integration layer *in its own language*.

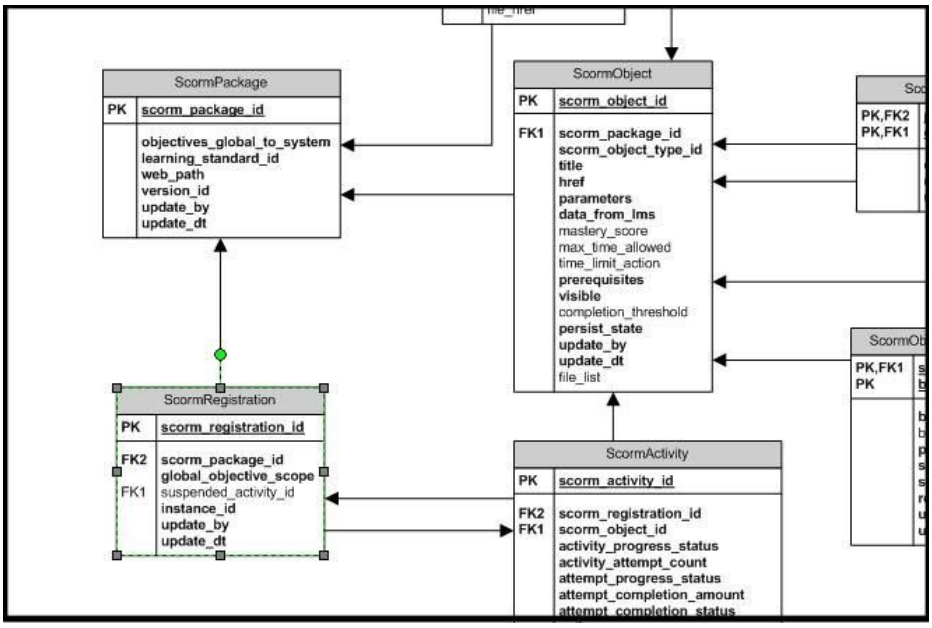
```
public abstract Learner GetLearnerInformation(ExternalRegistrationId externalReg, ExternalConfigura
    External Ids Passed into the integration layer
public abstract void RollupRegistration(ExternalRegistrationId externalReg, ExternalConfiguration e
```

These classes are each instances of the abstract ExternalId class which defines serialization methods for these classes. The serialization common to all external identifier objects allows LMS's to manipulate their identifiers as strings at times instead of instantiating actual ExternalId objects.

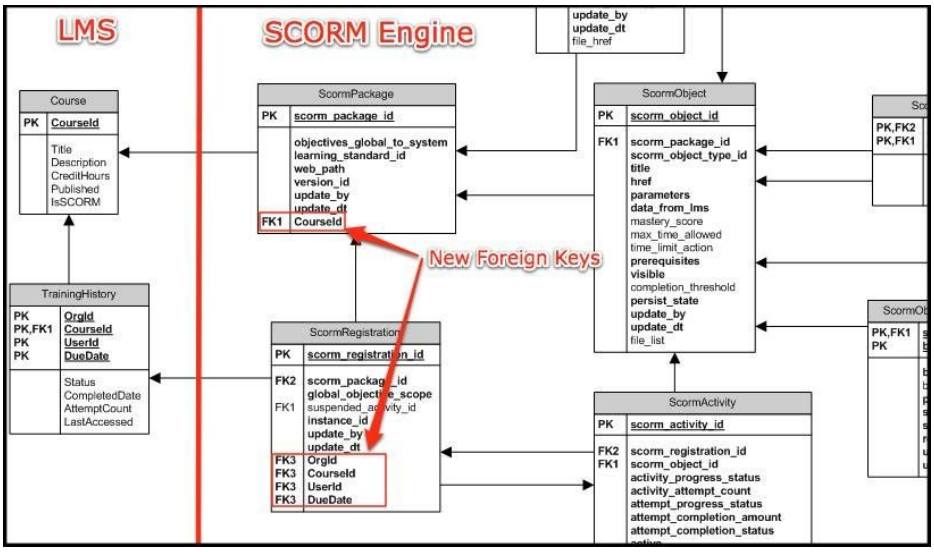
The relationships between the SCORM Engine's internal concepts and package and registration with the host LMS's associated concepts is also reflected in the database. In

keeping with the goal of tight integration with loose coupling, we allow two of the SCORM Engine's database tables to be modified during the integration. Both the ScormPackage table and the ScormRegistration table will have additional foreign key fields added to them to reflect their relationships with tables in the host LMS (tight integration). The other SCORM Engine tables remain untouched (loose coupling).

SCORM Engine database tables before integration



SCORM Engine database tables after integration



External Configuration

There is one more integration object that follows the same pattern of composition as the external package id and external registration id. The external configuration object is a

"tunnel" for passing information from the host LMS to the integration layer. The external configuration object is perhaps best explained with an example.

Take the SCORM Engine integration function LogError that was previously mentioned. This function is invoked by the SCORM Engine in the event of an unexpected runtime error so that diagnostic information about the error can be recorded for later analysis. Say Client X has service level agreements (SLAs) with a few select customers that imposes financial penalties for any system downtime. Because of these SLAs, Client X wants all of its support and development staff to be immediately notified by cell phone, pager, email, text message, singing telegram and carrier pigeon whenever an error affects a client with an SLA (note, we do not endorse inhumane treatment of pigeons). For all other clients, there's no need to interrupt anybody's sleep, so the error should just be recorded to a system log for later analysis.

To implement the LogError function in the integration layer, we need to have some information about the current client available to us to know what actions to take. This need poses a problem because it is the SCORM Engine that invokes the LogError function, not the host LMS. The SCORM Engine only knows about packages and registrations, not LMS client SLAs. To expand this problem out further to all clients and all integration functions. There are innumerable data points upon which the integration functions might rely to make decisions. The SCORM Engine can't possibly be aware of all of these options, so another solution is needed.

Enter the external configuration object. When the host LMS passes control to the SCORM Engine, it has the opportunity to pass along an external configuration object. Just like the other external ids (external package and external registration), the external configuration may contain any arbitrary set of properties. In other words, it can contain whatever information the integration layer might need. Anytime the SCORM Engine calls an integration function (i.e., anytime it might be calling back to the host LMS), it passes that same external configuration object to the integration layer. In this way, the external configuration object is like a tunnel that allows the host LMS to pass information through the SCORM Engine to the integration layer.

In our example above, the integration would define a property called IsSLACustomer in the ClientXExternalConfiguration class. Then, when launching the course, the host LMS would set this flag appropriately before handing control over to the SCORM Engine. The SCORM Engine would then save this configuration information and pass it to the integration layer every time an integration call is made. The integration function can then examine this flag and take the appropriate course of action in the event of an error.

SCORM Engine Integration

Last Modified: April 24, 2013
Current SCORM Engine Version: 2013.1

Welcome

Thanks for purchasing the SCORM Engine. We're eager to get started and to help you use the SCORM Engine to its full potential. This document will provide you with a road map to the integration process. It is not intended to be a comprehensive document listing every bit of functionality that the SCORM Engine provides, that would kill too many trees. Rather, this document will orient you to the integration process, set expectations and provide you with the key information needed to complete your integration.
If at any time you find yourself wishing that you had more information, or that the SCORM Engine could do something more, or that the integration could be handled differently, please ask. Chances are that the answer is "Yes! The SCORM Engine is built for that and here's how to do it". You've purchased a very flexible piece of software that can handle most anything that's thrown at it, and if it can't, we'll find a way to make it.

Working Together

So far, you've probably been working with Tim (our one-man sales department) to gain some familiarity with the SCORM Engine and have concluded that the SCORM Engine is the right solution for you. Together you've been through demonstrations, some technical discussions and have executed a contract for licensing. Now it's time for the business people to step aside and hand things over to the technical folks to work their magic.
We have a team of developers that handle SCORM Engine integrations. We will assign one of them to this project to act as your integration consultant. The integration consultant is there to walk you through the process step-by-step. The consultant will handle all of the necessary SCORM Engine customizations and guide you through the changes that need to be made in your LMS. Our integrators are quite knowledgeable and are there to answer any questions you may have during the integration phase.
During integration, we use a tool called Basecamp for project management. Basecamp provides a simple interface for exchanging messages, transferring files, tracking to-do lists and setting milestones. We strongly encourage the use of Basecamp for all electronic communication (you'll even notice our implementers logging summaries of phone calls in there). Basecamp provides you and us with a single place to go to find the latest deliverables, see notes from prior conversations and refresh our memories as to why things are implemented the way they are. We have found this tool to be invaluable to both our implementers and our clients. You will receive a welcome message via email with your log in information to Basecamp. We can add as many users as needed to the system, so if you have additional people who will be participating in this project or just want visibility into its progress, we'll be happy to give them access.
Our expectation is that the integration consultant will be working very closely and very intensely with your developers over the next few weeks. There is a rough project schedule listed below that represents the typical timeline for SCORM Engine integrations (this work can go considerably faster for simple integrations). There is work to be done both on our side and on yours. If this schedule doesn't match your expectations or if the resources on your side aren't fully available during this time period, please let us know so we can schedule accordingly.

SCORM Engine Integration Timeline
Week 1: Kickoff Meeting. Identify unique requirements. Generate integration layer. Initial deploy to client sandbox.
Week 2: Importing and Launching SCORM courseware.
Week 3: Rollup results. Coding for unique requirements.
Week 4: Skinning the player. Final tweaks. Testing and cleanup.

We have carefully architected the SCORM Engine to isolate our code from your code and vice versa. We maintain this barrier to ensure that changes to one system don't require additional integration work and don't adversely affect the other system. Similarly we find it best to maintain a similar boundary in the work that our integration consultants do and the work that your developers do. We are very reluctant to make changes or affect your code in any way. Your developers are the experts in your code and they are the ones that should be trusted to modify your system. We will work side by side with them, guide them and advise them as much as needed, but at the end of they day, they will be responsible for maintaining your system and they need to fully understand everything that is in there. Likewise, we do not expect your developers to become experts in our system overnight. We will gladly handle the the customizations and configurations needed in the SCORM Engine. If you prefer, we can also set your developers up with a simple development environment where they can make changes to the integration code. We are also happy to help your developers learn the innards of the SCORM Engine's source code should they be so motivated, but we don't want this learning curve to stand in your way.

The Kickoff Meeting

The first step in the integration process is a kickoff meeting with all involved parties. This is our chance to make introductions, work out some logistics and get the ball rolling. This is very much a working meeting from which we hope to take away most, if not all, of the information we need to generate your custom SCORM Engine integration.
The biggest part of the kickoff meeting is a tour of your LMS. We need you to show us around and give us a feel for how your LMS works. During the tour we will be looking for any unique requirements you have that might necessitate an advanced integration or other tweaks to the SCORM Engine. We've seen more than a few LMS's in our days so we will probably be very quick to understand yours.
We don't need to see everything your system has to offer, the main thing we need to figure out is how the entities in your system map to the entities in the SCORM Engine. Specifically, all LMS's have two entities that we will need to relate to, "packages" and "registrations".

> A "package" is often called a "course", "lesson", or "task". It is "the thing a learner takes". A package is the unit of online instruction that is registered for, launched and tracked. It corresponds to a single SCORM course.
> A "registration" is often called an "assignment", "instance", or "attempt". A registration is an instance of a user taking a package with a single set of tracking data.

If something just clicked and you see how these concepts map directly to your system, great! If not, don't worry, our consultants excel at comprehending your system and identifying the appropriate touch points.
During the tour, it will help to look at these areas of your LMS:

- How you import or create a new course.
- How a user is assigned to or registers for a course.
- How a user launches a course and sees his/her results.

How administrators view reports on the results of training

The LMS tour will segue into a look at your database schema. In the database schema, we are looking for two things, unique identifiers for a "package" and unique identifiers for a "registration". Every LMS has these concepts, but they can be called by different names and structured in different ways. These identifiers are the primary input our integration consultant needs to generate the first deliverable.

Information about the platform(s) we will be working with is the final piece of information we need from the kickoff meeting. Would you like a Java or .Net version of the Engine? Which database platforms do you need to support, SQL Server, Oracle, MySQL, others? The supported versions of these platforms are also helpful.

If there's time (and energy) during the kickoff meeting, we might get into topics that answer questions for later in the integration process. Specifically, the SCORM Engine needs to directly exchange two pieces of information with your system. First, the SCORM Engine needs ask your LMS for some information about each user (first name, last name and unique identifier). Second, the SCORM Engine needs to tell your LMS about the results of training it has delivered to a learner (we call this process "rollup"). We need to figure out the best way to perform this communication with your LMS. The SCORM Engine is quite flexible and can use any number of communication protocols, such as:

Direct database reads/writes
Web service invocations
API calls to existing system objects
Reading/Writing information from/to a querystring parameter

Kickoff Meeting Checklist

Introductions made and contact information exchanged
LMS tour
Unique identifier for package established
Unique identifier for registration established
Code and database plaftform(s) established
Communication protocol established (Optional - often discussed later)

The Setup Phase

After the kickoff meeting, we're going to give you some homework to do while we go off and generate the foundation of the integration.

Your homework is to get the SCORM Engine up and running in your development environment. The SCORM Engine itself isn't very usable without an LMS, so to get you started, we ship a simple SCORM Engine Console. The SCORM Engine Console is nothing more than a simple interface to the SCORM Engine that allows you to import and launch courses along with some debugging and system health tools. It will allow you to get everything set up and running in your environment before the integration with your LMS is completed. Once the integration code has been generated, the SCORM Engine Console provides us a way to deploy and test the integration code before it is tied into you system. See Instructions for Deploying the Noddy LMS for .Net and Instructions for Deploying the Noddy LMS for Java to get started.

While you are deploying the SCORM Engine, our integration consultant will be busy generating a customized integration for your LMS. This integration will be tailored to the unique identifiers and supported platform(s) we identified during the kickoff meeting. See

SCORM Engine Integration Architecture for more information about the technical aspects of this generated integration.

The Integration Phase

Now it's time to start hooking the two systems together. Our integration consultant will deliver a generated integration to you and instructions for deploying it to the Noddy LMS. The Noddy LMS will then be running with your specific code to let us simulate actions your LMS will eventually initiate after integration is completed.
There are three primary touch points where we need to integrate our systems, "import", "launch" and "rollup". This document will cover these touch points at a high level.

Key Integration Points

Import - The act of adding a SCORM package to your LMS. This is the place in your system where new courses are created or ingested.
Launch - The place where delivery of an online course is initiated by the user.
Rollup - The transfer of course progress data from the SCORM Engine into your LMS.

Import

We typically begin the integration process with the import mechanism. The goal of this part of the integration is to ensure that your LMS has an interface to upload and import SCORM conformant courses. Your LMS may already have an existing interface for importing external courses. If so, it is usually best to make slight modifications to the existing interface rather than attempting to create an entirely new interface, but that will vary from system to system.
There are three main outcomes that need to be met for the import integration to succeed:

File upload and deployment - SCORM courses are delivered as a set of files (often in a zip package). These files need to be uploaded to your LMS server and deployed to the appropriate locations for serving. The deployment process can be manual or automated, but there needs to be some form of administrative interface to enable it.
Invoking the SCORM Engine's import routines - The SCORM Engine has some routines that need to be invoked to discover the course and properly populate the SCORM Engine's tables with data about the course.
Package entity flagging - There needs to be some mechanism for flagging the package entity in your system as being a SCORM course that should launch with the SCORM Engine.

The SCORM Engine comes with some reusable interfaces that will handle the first two items above. These interfaces can be easily dropped into your existing interfaces. Alternatively, if these interfaces aren't an ideal fit, we can show you how to create your interface to invoke the SCORM Engine import methods through either web service calls or through direct API calls.
When thinking about the import mechanism, you will also want to think about package versioning. Package versioning controls how you handle updates to courses. We can help

you select from several built in schemes for dealing with versioning that the SCORM Engine offers. These schemes should allow us to mirror the versioning functionality that your LMS currently uses or they can be completely transparent to the LMS and only applicable inside the SCORM Engine.
The SCORM Engine offers over 60 customized settings for controlling how each courses is delivered to the user. We call these the "package properties". The ability to manipulate each course's package properties is essential to ensuring broad courseware compatiblity. The SCORM Engine offers a reusable interface for editing package properties (we recommend using this interface instead of your own as we are constantly adding new properties). After a course is imported, we need to make sure that your LMS provides administrators with a way of accessing these property settings.

Launch

Launching a course in the SCORM Engine is a simple matter of redirecting the user's browser to an appropriate URL with some querystring parameters appended. These launch parameters tell the SCORM Engine which course to launch, which registration identifier to associate the tracking data with and what "mode" to launch the course in. The format of these parameters is specific to your integration, however since the Noddy LMS is configured for your integration, it can provide examples of how to construct the launch settings.
When building the launch mechanism, we will want to consider the different modes in which content can be launched and how they map to the functionality in your LMS. For example, your LMS might provide a way to preview content or a way to review completed content. The SCORM Engine can handle these and other launch modes once they have been mapped to the functionality in your LMS.
Also during launch development, we will want to consider registration "instances". An instance is to a registration as a version is to a package. We will need to examine your LMS's policies around re-taking courses to see how they map to the SCORM Engine's registration instance schemes and then select the scheme most appropriate for your situation. Registration instances are closely related to package versions as often, new versions of packages will trigger new instances of registrations.

Rollup and Reporting

The final major integration point is rollup and registration. This is where we take all of the detailed data stored by the SCORM Engine for a particular registration, consolidate it down to the data that is relevant to your LMS and push the data into your system. The first step in the rollup integration process is determing what data you actually care about. Usually, most LMS's will want to know high level data about the course such as its status, score and the amount of time the learner spent in the course. The SCORM Engine can provide this and much more. The key is to figure out what your LMS needs to operate and getting that data in the right place. We will want to look at things such as the data that is displayed to the student, the data that is available to administrators via reports and the data points that trigger actions in the system (such as moving a course to the transcript or taking it off the learner's to-do list, etc). There will also be some business rules to flesh out, such as if a course is completed and failed, can the user retake it?
Once we have the required data identified, we then need to figure out the best way to technically get it into your system. Every time new data is saved to the SCORM Engine (this happens constantly while the course is being delivered), it triggers a process called "rollup".

We can configure this rollup process to take any action we need it to. For instance, we can have it write data directly into your LMS's tables, we can have it call a web service or we can make an API call into your system. The critical data that the learner sees and that triggers actions in your LMS is pushed to your system via the SCORM Engine whenever there is new data. If your system requires more detailed data for reporting, it can either be pushed with the summary data, or pulled on demand by a later process.

Further Integration Considerations

There are a few other things that need to be considered when completing a basic integration.
Learner information - The SCORM standards require that the SCORM Engine make some information about the learner available to the content. Specifically, we will need to figure out how to retrieve the learner's name and a unique identifer for the user from your system.
Database deployment - The SCORM Engine requires a database to operate. It can run on its own database, or within the context of your existing database. How this deployment is handled is largely a matter of style and your personal preference.
Code integration - Similar to the database, the SCORM Engine can be tightly integrated into your code base to be compiled together, or it can be run as a stand alone compiled application (potentially on its own server). How code is integrated and deployed is also largely a matter of your existing setup and procedures.
Skinning - The SCORM Engine is fully skinnable and can be customized to match whatever aesthetic scheme you desire.
Advanced Integration - There is much more that the SCORM Engine can do and many more ways in which it can be tightly integrated into your LMS. An integration may want to explore other areas like distributed content delivery, tight authentication, integrated error logging, partitioned databases, advanced importing or offline deliver (using our SCORM Untethered product which is sold separately). Your integration consultant will happily talk you through these areas.
Localization - The SCORM Engine capable of rendering the player and the SCORM Engine Package Properties Editor in a variety of languages. By default, localization is automatically based on the browser's configured language. However, if your LMS has its own means of establishing the user's locale the SCORM Engine can base its language off that instead.

Testing Phase

Once the integration is completed, it is of course important that we validate and test it. The best way to test the integration is simply to run a few sample courses through the cycle of importing, launching, and reporting. It is generally not necessary to test every combination and permutation of course type because the subtle errors that might be generated by course variations happen in the SCORM Engine itself and don't vary between integrations. To fully validate your SCORM conformance, ADL offers several Conformance Test Suites (one for each version of SCORM) that will thoroughly test your LMS and allow you to officially declare yourself SCORM conformant. It's not a bad idea to run these test suites, but generally not necessary to validate that your integration is functional.

Going Forth

Material Completion

Once you are able to import and deliver and rollup data from courses (even just a couple of examples that we provide), you have achieved what we refer to as "material completion". This a relevant milestone from both a process perspective and a contractual one. From this point forward, we have found that your requests are often better managed via our support portal (see below for information). Our project managers will confirm with you that you are comfortable importing content and that you can access the support portal as needed.
It is important to understand that *moving from the "implementation phase" to the "support phase" has no impact whatsoever on the level of support or access to our people*. It is merely a change in process that helps us take better care of you.

Certification

ADL offers a certification program that formally certifies or declares products to be SCORM conformant. The SCORM Engine has been certified for every version of SCORM, but unfortunately this certification does not transfer to your product. To be formally certified by ADL, you must put your LMS through the certification process. The process is not hard and we will be happy to walk you through it. It costs about $2,000 and gives you the right to say that your LMS is ADL SCORM Certified and to use the certification logo. We recommend that all of our clients get certified.

"Powered By" Logo Use

We want to make sure that you're getting the most out of SCORM Engine. Some of our customers prefer to tuck their use of our products away, and others want to scream from the mountaintop that they're using the best SCORM conformance software available. For the screamers, we've created a way for you to do just that. Visit our "Powered By" page for more info.

Support Process

We are always here for you, even after your integration is complete and your application is deployed. We have a dedicated support staff. If something goes awry after your integration is completed, please email us at support@scorm.com or visit our support portal. This will open up a support ticket and ensure you the fastest response. Our integration consultants rotate between many projects, including development of our products, and may not always be available to answer your questions directly once the integration is complete. Our support staff has unfettered access to all of our consultants and developers and can quickly put you in touch with the best person to resolve your problem.
For more details on the support process and our support portal, visit this document.

Troubleshooting

Nobody's perfect, we all make mistakes and things don't always go as expected. When problems arise, the SCORM Engine provides a few mechanisms for getting additional diagnostic information.

The most common problem our customers face is content behaving in unexpected ways. In almost every instance, this problem stems from a misunderstanding of the SCORM standard on the part of the content author, but we want to hear about these problems anyway so that we can ensure the SCORM Engine does everything it can to accommodate these varying interpretations of the standards. To diagnose SCORM content problems, the SCORM Engine maintains a very detailed debug log that tracks all of the SCORM calls made by the content as well as the internal SCORM logic that the SCORM Engine executed. This debug log can be accessed by clicking anywhere in the SCORM Engine's interface (the frames with the blue background in our default skin, or the frames that contain the table of contents or navigational elements). Then press the question mark key five times. This should cause the debug window to pop up. If there doesn't seem to be much information in the log, check the package properties for the course in question. The package properties have a few settings that control how much debug information is recorded, make sure that all of the properties are set to record information. More details about accessing the client-side debug log can be found in our support portal.

For deeper problems that affect the operation of the SCORM Engine itself, we have a detailed server-side log that can be accessed.

Rustici Software offers another tool that can be invaluable in diagnosing content problems. The SCORM TestTrack is a freely available hosted version of the SCORM Engine that is designed to quickly evaluate and debug courseware. The SCORM TestTrack always contains the latest updates and patches to the SCORM Engine. If content is not behaving as expected in your LMS, it is often useful to run the content through TestTrack as well to see if the problem is with your LMS and integration in particular or if it is a more general problem with the content or SCORM Engine.

Our clients will often instruct content vendors to validate content on SCORM TestTrack before attempting to import it into their LMS. This step can save countless hours of troubleshooting and messaging back and forth. We provide this free service for this very reason and we encourage you to take advantage of it. Some clients have also installed privately branded versions of TestTrack that are specific to their LMS. These licensed TestTrack instances can be customized to integrate directly into your content acceptance workflow to handle things like validation and approval.

Updates and Patches

We are constantly developing and improving the SCORM Engine. Our release schedule is largely dictated by the evolution of the standards, but we typically target about one major release per year. In the interim, we will periodically issue patches to fix significant errors or to deal with significant standards issues. These updates are available to any customer that is current with their licensing fees. Our support representatives will notify customers of new releases and we will post announcements to our blog as well. Patches are typically only applied as necessary to avoid overly burdensome update processes. Updates are generally straightforward to apply, but our consultants are available to you as needed.

Synchronized Code Bases

We maintain a current copy of the integration code specific to your LMS in an internal source control system. This system allows us instant access to your specific code base if we

need to reference it to help troubleshoot an issue or upgrade your system. If you make any changes to your integration, please send them our way so we can keep our copy up to date. Also, if you need to make any changes to the source code of the core SCORM Engine, please let us know so that we can try to work your requirements into a release and keep you on the standard maintenance path.

SCORM Engine Recommended Requirements

SCORM Engine Version: 2013.1

Last Updated: July, 2013

The SCORM Engine is designed to be deployed in a wide variety of configurations. Listed below are the basic requirements for using the SCORM Engine.

A web server capable of running server-side code. Currently we support:

- Microsoft IIS – must be configured to run ASP.NET using the .NET runtime version 3.5 or higher
- A Java application server (such as Apache Tomcat, WebSphere®, JBoss® and WebLogic®) running J2EE 1.4 or higher with J2SE 5.0.

A relational database. Currently we support:

- SQL Server 2005 and higher
- Oracle 10g and higher
- MySQL 5 and higher
- Postgres 8.4 and higher
- No database, simply persist information in XML files (not recommended for production web environments)

A computer – Your hardware requirements will vary greatly depending on many factors, most notably your expected user base and configuration from above. In our development environment, we have tested on the following "minimum" system and the SCORM Engine behaves normally under small load. See this document for information on scalability.

Windows 2000 Server SP 4

Intel Pentium 4 1.8Ghz CPU

256 MB RAM

30 GB Hard drive

For development and integration, it is often necessary to be able to view and modify source code. When working with .Net it is helpful to have Visual Studio.Net 2005 installed (with SP1 and the web application project type enabled). When working with Java, it is helpful to have Eclipse 3.2.2 or higher. Should you not have access to these tools, our consultants can usually create any necessary code on their computers.

Other helpful items

Administrative access to the servers is often quite helpful in resolving any permissions related issues after deployment

For remote installations (or for technical support after an onsite installation), access to screen sharing software greatly simplifies matters. Rustici Software can provide this software, however, for it to be useful, we need to ensure that your firewall allows it to operate.

On the client side, all that is required is a web browser. The SCORM Engine supports all modern web browsers and we constantly strive to stay up to date with new browser versions. No plug-ins are needed, nor is Java support required. When using Microsoft Internet Explorer before IE 7, ActiveX controls need to be enabled to facilitate the user of the

XmlHttp object. Note, the XmlHttp object is not an custom ActiveX control, but rather an object built into all modern browsers.

Package Properties Reference

Please see this .pdf for the package properties reference

Notable New Features

Tin Can API

Version 1.0.0 of the Tin Can API specification has been implemented within the SCORM Engine. If you are not familiar with the Tin Can API, please visit http://tincanapi.com.

Offline/Mobile Player API

The SCORM Engine now has an offline/mobile API exposed which contains libraries in objective C and java as well as a server-side component. The latest release of the SCORM Engine contains the server-side portions of this API along with supporting integration methods. To learn more about how the offline player works with SCORM Engine, please visit http://scorm.com/scorm-solved/scorm-engine/mobileoffline-scorm/

Internet Explorer Compatibility Property

When content renders within the SCORM Engine player frameset, it will be rendered using the Internet Explorer rendering mode of the parent frame. There is some content which does not work properly in the latest standards mode of IE. As such, the SCORM Engine previously set a META tag to force IE8 Compatibility mode. The problem is that there is newer content which wants to take advantage of the latest html5 rendering modes. As such, we have introduced a new package property for IE Compatibility which can be accessed via the Package Properties Editor. You can also set the import-time default for this through an integration method. The default is EmulateIE8 Compatibility mode, just like it was hard-coded in the previous version of the SCORM Engine.

Error Fixes

Upgraded third-party javascript libraries, including jQuery and Sarissa to latest versions to stay current with browser compatibility.
Handle long AU titles for AICC by truncating to 200 chars.
Improved performance where global objectives are broadly used - new index.
Fixed handling of non-print characters in suspend_data.
Console improvements including the complete utilization of external configuration.

SCORM Engine Launch Parameters

SCORM Engine Version: 2013.1

Last Updated: April 24, 2013

When launching the SCORM Engine, there a several parameters that can be passed to it via the querystring. These parameters tell the SCORM Engine which course to load, how to track the learner's progress and how the course should behave.

Parameter Name	Possible Values (should be URL encoded)
"configuration"	A serialized external configuration object
"registration"	A serialized external registration id
"package"	A serialized external package id
"manifestDirPath"	A valid file path or HTTP path
"webPath"	A value HTTP path to a directory
"tracking"	"true" or "false"
"forceReview"	"true" or "false"
"regForCredit"	"true" or "false"
"cc"	CultureCode to choose a delivery language. (e.g., 'en', 'fr'. Note, this functionality is not turned on by default.)
"startSco"	An Item Identifier that identifies a SCO in the manifest.

Configuration

The "configuration" parameter contains a serialized version of the specific integration's external configuration object. This parameter is always required to be present, but usually does not have to contain a value. The external configuration object is used to vary the behavior of the SCORM Engine in the integration layer. Passing in a string representation of this object at launch, will cause an instance of the specific integration's external configuration object to be instantiated and passed into the integration layer whenever an integration function is called.

Registration

The "registration" parameter contains a identifier that should be associated with the SCORM tracking information for this course launch. This is the "external registration id". The format of the registration parameter should be a serialized version of the specific integration's external registration id object. If the external registration id specified in this parameter does not already exist in the SCORM Engine, then by default a new registration

will be created (although this behavior can vary based on the integration layer and the "CreateRegistrationIfNeeded" SCORM Engine setting). If the specified external registration id does exist, then that registration will be resumed and the tracking data from any previous attempts will be restored.

Package

The "package" parameter contains an external package id identifying a package that has already been imported into the SCORM Engine. If no "registration" parameter is passed in, then the package identified in this parameter will be launched in a preview mode with no tracking. If a "registration" parameter is passed in and a new registration needs to be created because the external registration id does not exist, then the package identified by this parameter will be associated with the newly created registration. If the "registration" parameter is passed in and there is an existing registration, then the "package" parameter is ignored.

Registration Parameter	Package Parameter	Action
Not Included	Included	Package is launched in preview mode
Included, no matching registration exists	Included	New registration is created with specified package
Included, matching registration does exist	Included	Existing registation is launched, package parameter is ignored. If the registration id does exist, the package parameter is not required.

ManifestDirPath and WebPath

The "manifestDirpath" and "webPath" parameters are used in conjunction with one another. They enable the SCORM Engine to launch a course that has not yet been imported. The course's manifest is parsed on the fly and the course is launched in a preview mode with no tracking. The "manifestDirPath" parameter should contain either a file path (accessible to the server on which SCORM Engine is deployed) or an HTTP location of the course's descriptor file (usually the imsmanifest.xml file). When launching a course that has not yet been imported, you also need to pass in the "webPath" parameter to tell the SCORM Engine where the course resides. The "webPath" parameter is an HTTP path to the root of the course (usually the directory where the manifest resides).

Tracking

The "tracking" parameter provides a way to launch a registration without saving any of the tracking data associated with the course. When the "tracking" parameter is provided

set to "false", the SCORM Engine will still accept all of the SCORM data sent to it by the content, but it will only persist it for the duration of the session. When the learner exits the course, all of the new data is discarded and the original state is preserved. This mode is useful for allowing learner's to review content that has already been completed to ensure that the record of their completion is not overwritten. If not included, the default value for this paramter is "true".

ForceReview

When set to "true", the "forceReview" parameter ensures that the data model element for mode ("cmi.mode" or "cmi.core.lesson_mode") is always set to "review". This setting is often used in conjunction with the "tracking" setting to provide learners an opportunity to review a course after it has been completed. If not included, the default value for this paramter is "false".

RegForCredit

The "regForCredit" parameter is used when the SCORM Engine creates a new registration upon launch. If the "regForCredit" parameter is passed in and set to "false", the SCORM Engine will create a new regisration with the data model element for credit ("cmi.credit" or "cmi.core.credit") set to "no credit". This setting is useful for lanching courses that should be tracked but that don't "count" for anything. If not included, the default value for this paramter is "true".

CC

The "cc" parameter can be used by a client integration to force the delivery language to a particular culture code, e.g., 'en', 'fr', via the launch string. This functionality is not enabled by default. To make use of this parameter the client should override the SetCulture() integration method in the integration layer.

StartSCO

If provided, this parameter identifies a SCO that the SCORM Engine should launch first. If not provided, the SCORM Engine will either launch the first SCO (for new registrations) or the SCO from which the learner suspended a previous attempt (for previously attempted registrations). Note that these default SCOs can be altered by SCORM 2004 sequencing rules in the content. The format of this parameter is a string representing the Item identifier associated with the SCO to be launched in the manifest. Note that if the manifest contains SCORM 2004 sequencing rules, it might not always be possible to launch the specified SCO (if for instance it's prerequisites are not met). In this case, the learner will be prompted with a message to make another selection.

Serializing and Encoding

All values must be properly escaped (or "URL Encoded") when they are included in the query string. It is important not to double encode the values. All common programming languages include a library function for properly escaping values to be placed in a

querystring. For static values that do not change, it can be helpful to use a tool to perform the one time encoding.

When passing an external package id, external registration id or external configuration id to the SCORM Engine these objects must be represented in their serialized state. The number, type and name of the properties contained in each of these of these objects in unique to each integration. Often, there is just one property, in which case the serialized version of the object is just the value for that property. However, in cases where there is more than one property, the serialized form of the object is a series of name value pairs separated by delimiters. By default, the delimiter that is between the name and value is a pipe character ("|") and the delimiter between a set of names and values is an exclamation mark character ("!"). Note that these defaults will vary based on the version of the SCORM Engine and can be different for each integration.

For example, if an external registration id is composed of two fields, userName and courseId, then a serialized external registration id might look like this:

userName|joeuser!courseId|42That value indicates an external registration id with a value for userName of "joeuser" and a vale for courseId of "42". When passing the serialized value into the SCORM Engine, the entire serialized value needs to be URL Encoded. Note that "!" does not need to be escaped and that the escaped represenation of "|" is "%7c". Once escaped, the above example would look like:

userName%7cjoeuser!courseId%7c42The Noddy LMS can simplify the process of created serialized and escaped object values. When creating new registrations, the Noddy LMS will display the proper launch URL. Since the Noddy LMS can be configured to use your specific integration, it is easy to simply copy and paste values into your code.

Common Configurations

Launch a registration "normally"

Parameter Name	Value to pass in
"configuration"	A serialized external configuration object if used by your integration.
"registration"	A serialized external registration id
"package"	A serialized external package id
"manifestDirPath"	Not included
"webPath"	Not included
"tracking"	Not included
"forceReview"	Not included
"regForCredit"	Not included

Launch a completed registration in review mode with no changes to the tracking data

Parameter Name	Value to pass in
"configuration"	A serialized external configuration object if used by your integration.
"registration"	A serialized external registration id
"package"	Not included
"manifestDirPath"	Not included
"webPath"	Not included
"tracking"	"false"
"forceReview"	"true"
"regForCredit"	Not included

Launch an imported course in preview mode with no tracking

Parameter Name	Value to pass in
"configuration"	A serialized external configuration object if used by your integration.
"registration"	Not included
"package"	A serialized external package id

"manifestDirPath"	Not included
"webPath"	Not included
"tracking"	Not included (only relevant if a registration is passed in)
"forceReview"	Not included
"regForCredit"	Not included

Launch a course that does not "count" for credit, but should still be tracked

Parameter Name	Value to pass in
"configuration"	A serialized external configuration object if used by your integration.
"registration"	A serialized external registration id
"package"	A serialized external package id
"manifestDirPath"	Not included
"webPath"	Not included
"tracking"	Not included
"forceReview"	Not included
"regForCredit"	"false"

Launch a course directly from a manifest that has not yet been imported

Parameter Name	Value to pass in
"configuration"	A serialized external configuration object if used by your integration.
"registration"	Not included
"package"	Not included
"manifestDirPath"	File path to manifest
"webPath"	Web path to course directory
"tracking"	Not included
"forceReview"	Not included

"regForCredit"	Not included

Mode and Credit

The SCORM runtime data model contains two elements that indicate the context in which a course was launched. This context is affected by the parameters that are passed into the SCORM Engine on launch. The "mode" data model element indicates that the course was launched either in a "normal", "review" or "browse" mode. The "credit" data model element indicates whether or not the course is being taken for credit.

Mode:

By default, when a course is launched with a registration id, the "mode" will be "normal".
By default, when a course is launched without a registration id, the "mode" will be "browse"
If the "forceReview" parameter is included with a value of "true", then the "mode" will always be "review".
Note: per the SCORM specification, "mode" can also change to "review" after a SCO is completed

Credit:

By default, when a course is launched with a registration id, the "credit" value will be set to "credit".
When a course is launched with a registration id and the "regForCredit" parameter is passed with a value of "false", the "credit" value will be set to "no credit".
When a course is launched without a registration id, the "credit" value will always be set to "no credit".

SCORM Engine Scalability

SCORM Engine Version: 2013.1

Last Updated: Aug 2, 2012

Introduction

Clients often ask us “How many users can the SCORM Engine can support?” Our answer usually falls somewhere between “a lot” and “it depends”. Both are true, but not very helpful. This document will shed some more light on the empirical data we have about the scalability of the SCORM Engine as well as the results of some measured stress testing we recently performed.

Why is this such a hard question?

There are many factors that affect the load on the server when delivering online training through the SCORM Engine. All of them can greatly impact scalability.

Deployment Variability

The SCORM Engine is designed to be tightly integrated into external LMS systems, every one of which is different. Most significantly, the LMS's we have integrated with use just about every application stack on the market. The SCORM Engine is deployed on Windows servers, Linux servers and even the occasional Mac server. It runs on top of SQL Server, Oracle, MySql, DB2 and a few other databases. These environments are sometimes replicated, sometimes clustered, sometimes load balanced and all of them have different authentication and security requirements.

Integration Variability

The SCORM Engine has a very flexible interface with which it ties into a client's LMS. How this interface is used and configured can have a significant impact in the server side load. For instance, the amount of data that is communicated and shared across systems will have a measurable impact on performance. The method in which this data is transmitted also comes into play; do the systems communicate via SOAP requests, through direct API calls, through access to a shared database or something else?

Course Variability

SCORM offers allows for a lot of flexibility in how courses are put together. There is a big difference in the amount of data that the SCORM Engine must track for a single SCO course verses a course with one hundred SCOs. Within each SCO, there can also be a huge variation in the amount of data that the SCO chooses to record and track. Some SCOs do nothing more than indicate that they are starting and completing while others will track the learners' progress in detail (including things like how they answer questions and how they are progressing on various learning objectives). How courses use SCORM 2004 sequencing and how large the actual courseware files are will also impact performance.

Usage Variability

Different communities of practice will experience different usage patterns of their LMS. Some communities will have users that take all their training in clumps while others will have users who only access the system in short bursts. Some systems are mostly accessed during business hours while others are active twenty-four hours a day. Systems that support supplemental material in a classroom may have many users all start a course simultaneously, while more asynchronous systems will have users starting and stopping throughout the day.

Empirical Evidence

Empirically we know that the SCORM Engine can scale quite well. Several of our clients operate very large LMS instances in which the SCORM Engine performs admirably. One

client in particular tracks over 1.5 million users and routinely processes over 50,000 course completions in a day. Other clients serve entire military branches from server farms distributed throughout the globe. Of course there have been occasional hiccups, but by and large the SCORM Engine handles these loads quite well.

Architecturally we designed the SCORM Engine for scalability from the start. One of the more significant architectural decisions we made was to push the SCORM sequencer down to the browser. Interpreting the SCORM 2004 sequencing rules can require a fair amount of processing. In a conventional SCORM player, in between every SCO, data must be sent to the server, undergo extensive processing and then be returned back to the client. In the SCORM Engine, all of this processing happens locally in the browser, eliminating a significant load on the server as the course is delivered. Typically the bulk of the server-side load happens when a course is launched as all of the required course data is retrieved from the database and sent to the browser. During course execution, incremental progress data is periodically sent to the server resulting in relative small hits to the server as this data is persisted to the database.

Stress Testing Results

In February of 2008, we conducted a performance test to get benchmark numbers reflecting the scalability of the SCORM Engine as represented by the number of concurrent users accessing the system. The intent of this test was to establish a benchmark of scalability on a simple representative system which can be used to roughly infer the performance of a more comprehensive system. As mentioned above, there are a number of variables that contribute to the scalability of a production system, any one of which can create a bottleneck or stress a system. We highly recommend adequate stress testing in a mirrored environment prior to deployment.

Methodology

To simulate user activity within the SCORM Engine, we began by selecting four diverse courses to use in our testing. The courses included:

A single SCO, flash-based SCORM 1.2 course
A short SCORM 2004 course that reports detailed SCORM runtime data to the LMS
A simple sample SCORM 2004 course that performs simple sequencing
An advanced SCORM 2004 course that makes extensive use of sequencing

We then captured the client-server HTTP interactions of a typical user progressing through each course. This data was massaged into a script that would accurately simulate many users hitting the system and updating their own individual training records.

Our test was set up on a dedicated server farm consisting of a single central LMS server and two clients from which the user requests were made using The Grinder load testing software. The LMS server has the following specifications:

Processor: Intel Pentium D 3.00 GHz

RAM: 2 GB

Disk: 130 GB

Operating System: Windows Server 2003 Enterprise Edition, Service Pack 1

Web Server: IIS v6.0 with ASP.NET 1.1.4322

Database: SQL Server 2005

SCORM Engine: Alpha version of 2008.1, configured to persist data every 10 seconds and rollup minimal data to an external system

The two client machines have similar specifications. If you're not a numbers person, you can think of it this way, these were the cheapest servers we could buy from Dell in the summer of 2007, with Microsoft software typical of the day. All machines were directly connected to one another on a gigabit switch.

Simulating concurrent users proved to be trickier than expected due to the need to stagger the start of each user simulated user's progress through the course. Our solution was to start the desired number of users at randomly spaced intervals over a period of 20 minutes. Since some users would complete their course in less than 20 minutes, each user was set to start the course again after completing it. After allowing 20 minutes to get up to full load, we measured system performance over the course of 10 minutes to get an accurate feel for how the system performed under load.

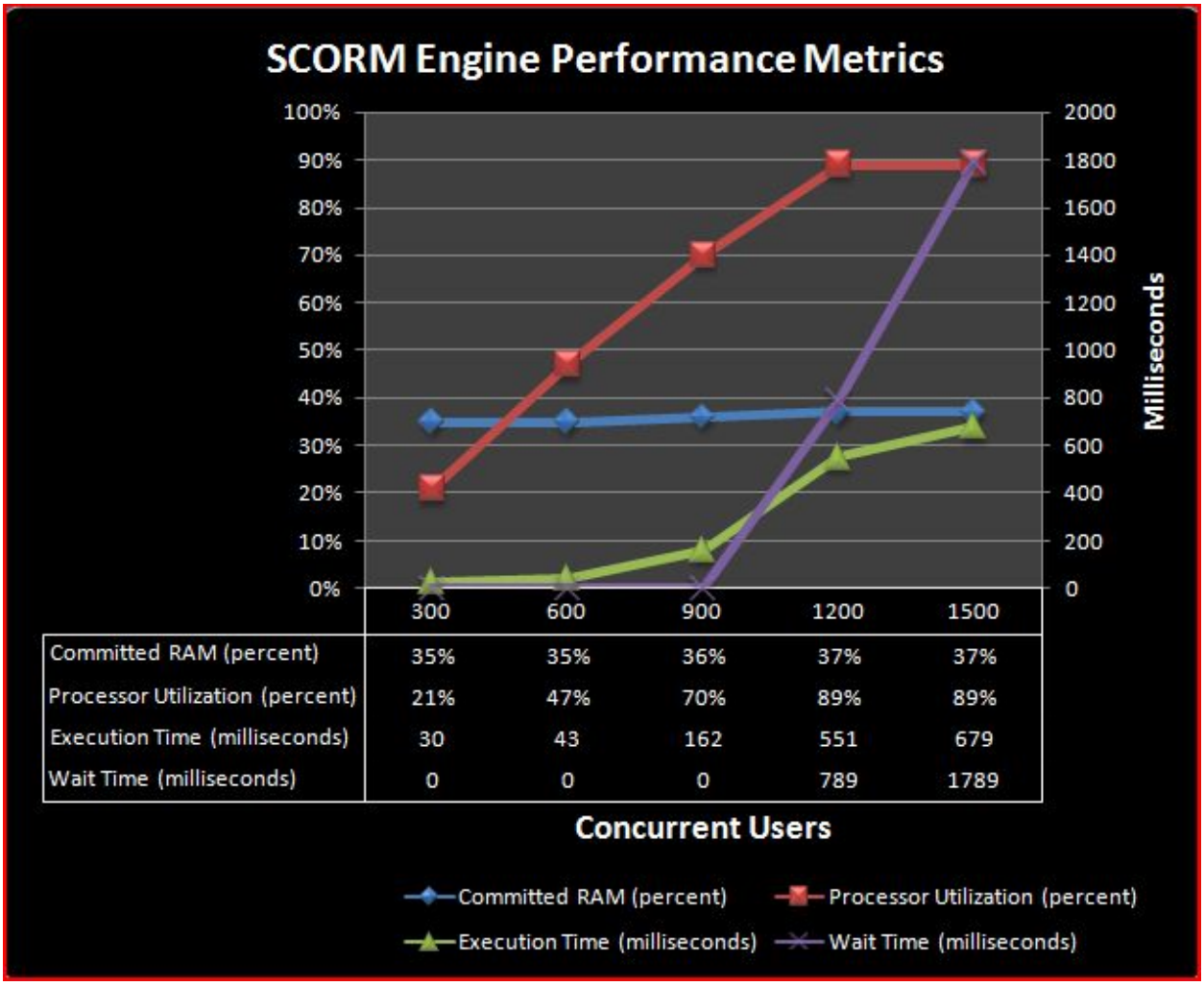
During that 10 minute period, we monitored the following metrics:

Processor Utilization – Percentage of available processor time used by the application
Committed RAM – Percentage of available RAM used by the application
Wait Time – The amount of time the HTTP requests waiting in a queue before they were processed.
Execution Time – The amount of time it took to actually process each web request once it reached the front of the queue

Note that we did not monitor bandwidth utilization. The reason for this decision is that typically the bandwidth consumed by the SCORM Engine pales in comparison to the bandwidth used by the actual training material. Thus we did not think bandwidth relevant to a discussion on the scalability of the SCORM Engine; however it could play a significant role in the scalability of a production LMS system.

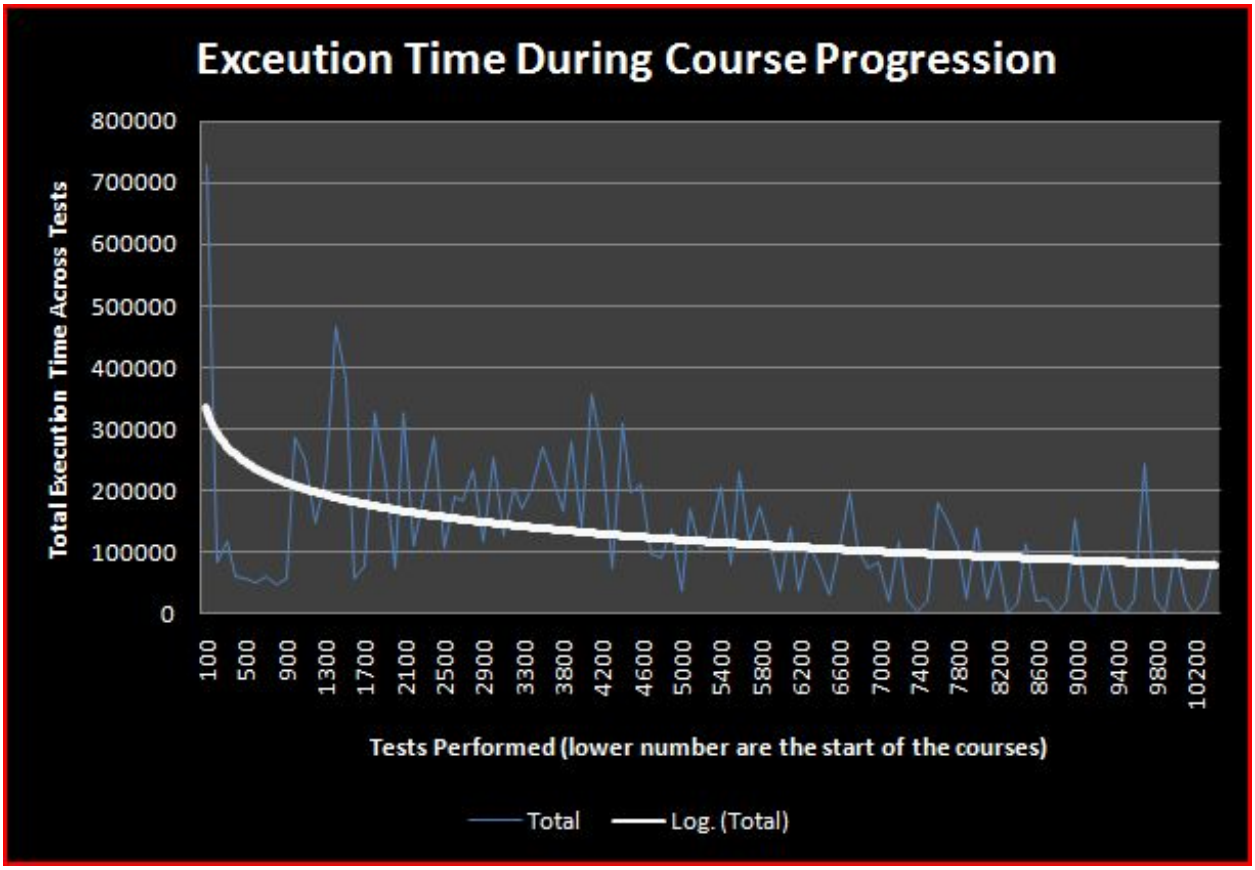
Results

Our intent was to run these tests and continually increment the number of concurrent users until either a resource was constrained or the average server response time exceeded one second. Both events seemed to happen around the same time at about 1000 concurrent users.



As you can see in the results above, processor utilization seems to be the constraining resource in this system configuration. There is a linear relationship to processor utilization and the number of concurrent users until the processor utilization is maxed out at around 90%. RAM utilization increases only slightly with load. Failure begins to occur as the processor becomes overwhelmed and HTTP requests begin to get stacked up in a queue waiting for the processor to become available.

We also analyzed the server load as users progress through a course. The logarithmic trend line in the graph below clearly shows the initial front end load (seen by the spike in the first request) followed by a relatively steady load as the course progresses.



Conclusions

A single server of modest horsepower can handle a concurrent user load of approximately 1000 users. Making some assumptions, we can get a rough idea of how many total system users this represents. Assume that a user will take a SCORM course once a week (probably an optimistic assumption). Assume that each SCORM course lasts one hour and that all training is evenly distributed during a 12 hour window each day. That means that each system user consumes one hour out of 60 available every week (assuming a 5 day week). If 1000 users can access the server at any given moment, we roughly have 60,000 available hours. Since each user consumes roughly one hour, theoretically this server could support an LMS with 60,000 registered users. Obviously these calculations are rough and don't allow for spikes in usage, but they at least provide an estimate from which to begin.

Update - July 2009

In version 2009.1 of the SCORM Engine we made a change to significantly improve scalability. Specifically, we removed the UpdateRunTimeFromXML stored procedure. This stored procedure handled the updating of all run-time progress data in one large database call. This increases the efficiency of each run-time update by reducing the number of individual database calls. Under light load, this procedure is rather efficient. Under very heavy load, this

procedure was found to cause resource contention, leading to slower performance and even deadlocks. What was designed as a performance optimization actually turned into a performance bottleneck, thus we have removed this stored procedure from the SCORM Engine. This change can usually be retrofitted into prior versions of the SCORM Engine by making a simple configuration change. Please contact us if you would like help changing your system.

SCORM Engine Settings

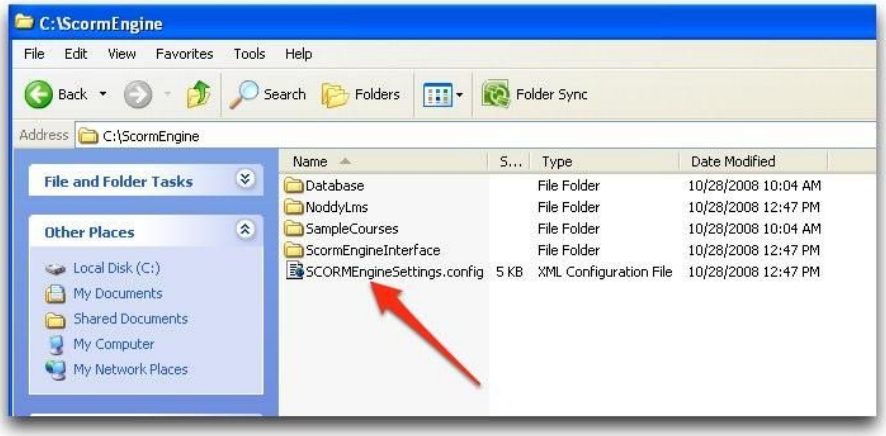
The SCORM Engine contains a number of configuration settings. These settings contain logistical information about how the SCORM Engine is deployed and they control how the SCORM Engine behaves. The SCORM Engine settings do not need to be changed frequently. They are typically only accessed during integration with another system and during deployment to new servers. If non-static values for any of these settings are needed, their values can be altered through the integration layer instead of being statically stored in the configuration file. The settings control the operation of both the SCORM Engine and of the Noddy LMS.

Working with the SCORM Engine Settings

The method for accessing and changing the SCORM Engine settings varies depending on the platform you are running (.NET or Java).

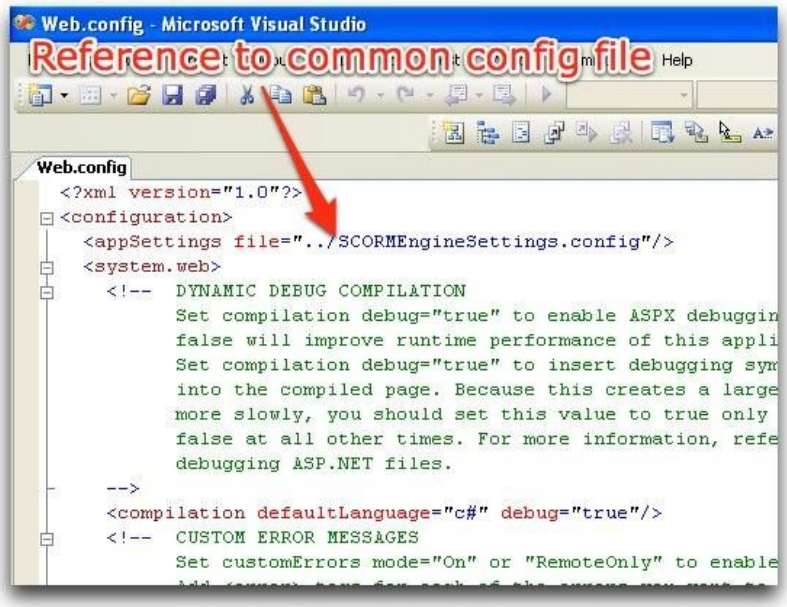
.NET

In a .NET installation of the SCORM Engine, the settings are contained in a file called "ScormEngineSettings.config". This file is located at the root of the SCORM Engine installation, in the directory above the "ScormEngineInterface" and the "NoddyLms" directories.



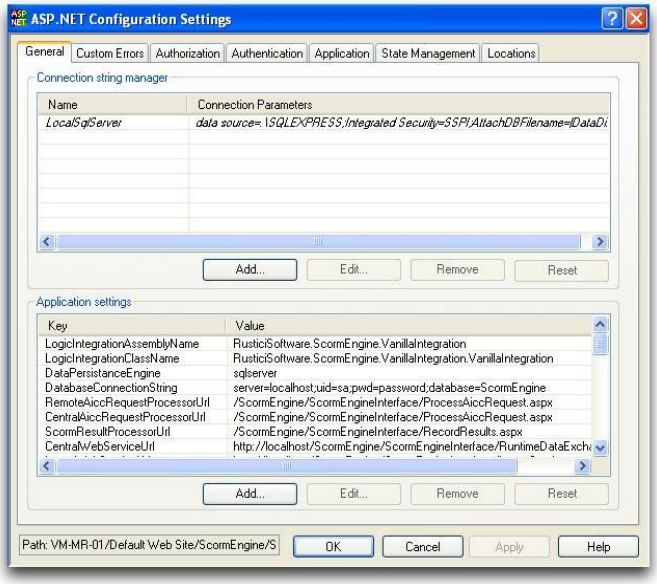
The "ScormEngineSettings.config" file is a standard ASP.Net Configuration file. It is included by reference in the "web.config" files in the ScormEngineInterface and NoddyLms

directories. The settings for the SCORM Engine can be stored in any valid and accessible ASP.Net configuration location.



The "ScormEngineSettings.config" file is a standard XML file that can be edited in any text editor or XML editor.

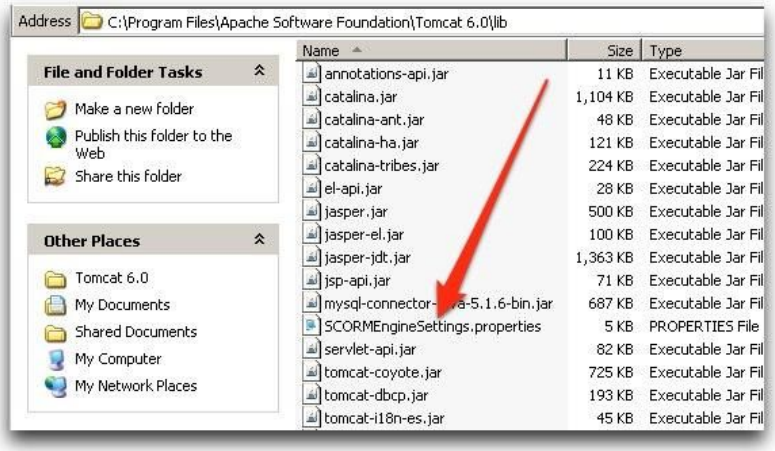
Alternatively, recent version of IIS include an "Edit Cofiguration" button on the ASP.NET tab of the applications properties. This button brings up a GUI for editing application settings individually.



IIS doesn't automatically detect changes made to the "ScormEngineSettings.config" file. In order to get the changes you make to be detected by IIS, you need to either: restart IIS or make a small change to both "web.config" files (one in the ScormEngineInterface directory and one in the NoddyLms directory) and resave them. IIS will pick up a change to the web.config files automatically. An easy way to get the changes picked up is to open the "web.config" files, type a character, delete the character and then re-save.

Java

In a Java installation of the SCORM Engine, the settings are contained in a file called "SCORMEngineSettings.properties". This file should be deployed to a location in the web applications' class path.



The "SCORMEngineSettings.properties" file is a standard Java configuration file. This file is a standard XML file that can be edited in any text editor or XML editor.

Depending on your Java Application Server, you may need to cycle the application in order for configuration changes to be picked up.

The Settings

The SCORM Engine settings can be broken up into eight groups:

- Integration class
- Data persistence
- URLs
- Upload import control
- Registration instance and package versioning
- Optional SCORM Engine features
- Debug settings
- Central / remote architecture

Integration Class

These two settings determine which class the integration factory will load. This class should be the concrete implementation of the integration interface that is designed to work with the current LMS (the integration layer).

LogicIntegrationAssemblyName - The .NET assembly in which the integration class resides. The assembly name is the name of the DLL containing the code and often corresponds to the namespace of the class in which the integration class resides. This setting is not required for Java.

Example: "RusticiSoftware.ScormEngine.VanillaIntegration"

LogicIntegrationClassName - The fully qualified name of the actual integration class to load. Usually, this is the assembly name concatenated with with the class name.

Example*:* "RusticiSoftware.ScormEngine.VanillaIntegration.VanillaIntegration"

Data Persistence

The data persistence settings control how the SCORM Engine accesses the database.

DataPersistenceEngine - The SCORM Engine supports many different data persistence options. This setting controls which of the supported options the SCORM Engine will use. The options are listed in the table below.

Value	Description
"compactsqlserver"	Connect to a Microsoft SQL Server CE database. (.NET Only)
"db2"	Connect to a IBM DB2 database. (.NET Only)
"db2_zos"	Connect to a IBM DB2 for z/OS (Mainframe). (.NET Only)
"mysql"	Connect to a MySQL database.
"odbc"	Connect to any database that supports an ODBC interface. (.NET Only)
"oracle"	Connect to an Oracle database
"oracle-not_optimized"	Connect to an Oracle database without using the bulk persistence stored procedure. [DEPRECATED in v2009.1 - use the "oracle" setting instead]
"ole"	Connect to any database that supports an OLE interface. (.NET Only)
"plugin"	Use a custom devloped data persistence mechanism.
"sqlite"	Connect to a SQLite database. (.NET Only)
"sqlserver"	Connect to a Microsoft SQL Server database.
"sqlserver-not_optimized"	Connect to a Microsoft SQL Server database without using the bulk persistence stored procedure. [DEPRECATED in v2009.1 - use the "sqlserver" setting instead]

DatabaseConnectionString - The connection string that the SCORM Engine will use to connect to the database. The value for this setting varied based on the platform you are running.
In .NET, the value is an actual connection string. The format of the connection string will vary depending on the data persistence engine selected. Some examples of connection strings on various platforms are included in the table below.

Data Persistence Engine	Example .NET Connection String
SQL Server	"server=localhost;uid=sa;pwd=password;database=ScormEngine"
MySQL	"Host=localhost:3006;UserName=root;Password=password;Database=ScormEngine;"
Oracle	"Data Source=oracledb.local;User Id=ScormEngine;Password=password;Integrated Security=no"
IBM DB2	"Provider=IBMDADB2;Data Source=ScormEngine;UID=ScormEngineUser;PWD=password"

In Java installations, this parameter is optional. If not value is provided, the SCORM Engine will use a pre-defined data source named "jdbc/ScormEngine". If you would like the SCORM Engine to use another named data source, simply include it's name as the value of this setting.

Advanced Data Persistence Settings

These settings are all optional and are only used used in advanced scenarios to accomodate varied data persistence options. The data persistence engine setting is essentially a shortcut to specify various combinations of pre-define data peristence classes and should be used instead of these settings unless an uncommon scenario is encountered.

DataPersistanceAssemblyName (optional) - When using the "plugin" data persistence engine, this settings specifies the name of the assembly from which to load the data persistence plug-in class. This setting is not required for Java.

Example: "RusticiSoftware.ScormContentPlayer.Logic"

DataPersistancePersistClassName (optional) - When using the "plugin" data persistence engine, this setting specifies the name of the class within the DataPersistenceAssembly that handles persisting data to the database. This class contains the actual SQL (or other commands) for manipluating SCORM Engine data within the database.

Example: "RusticiSoftware.ScormContentPlayer.Logic.SqlPersistImplementation"

DataPersistanceRetrieveClassName (optional)- When using the "plugin" data persistence engine, this setting specifies the name of the class within the DataPersistenceAssembly that handles retrieving data from the database. This class contains the actual SQL (or other commands) for manipluating SCORM Engine data within the database.

Example: "RusticiSoftware.ScormContentPlayer.Logic.SqlRetrieveImplementation"

DataHelperAssemblyName (optional)- When using the "plugin" data persistence engine, this settings specifies the name of the assembly from which to load the data helper plug-in class. This setting is not required for Java.

Example: "RusticiSoftware.ScormContentPlayer.DataHelp"

DataHelperClassName (optional) - When using the "plugin" data persistence engine, this setting specifies the name of the class within the DataHelperAssembly that handles connecting to and interacting with the database. This class contains generic helper functions that abstract the process of connecting to an querying a specific database.

Example: "RusticiSoftware.ScormContentPlayer.DataHelp.JdbcDataHelper"

DataPersistanceUseStoredProcsIfAvailable (optional, default="true") - This setting modifies the behavior of the SQL Server and Oracle data persistence engines to use stored procedures or regular SQL when performing some functions. In general, this setting should not be modified. Possible values: "true" or "false".

UseGuidAsObjectId - This setting allows the SCORM Engine to toggle between using integers and GUIDs as primary keys in its tables. Possible values: "true" or "false".

DBMaxStringLength - This optional setting sets the maximum string length to allow through to be persisted in the database. This applies to all strings passed to the database, including text fields.

DatabaseTimeout - Value used by the default integration implementation specifying the timeout threshold for database operations. In seconds.

DatabaseSchema - String value used by the default integration implementation specifying the schema name that should be prepended to table names.

URLs

The URLs specified in the SCORM Engine settings contain information about where the SCORM Engine was deployed and where it should make requests to. These settings allow for

a lot of flexibility in how the SCORM Engine is deployed, in most situations however, they will all point to the single directory in which the SCORM Engine is deployed.
Unless otherwise specified, the URLs can either be fully qualified (ex: "http://www.mylmsserver.com/ScormEngine/pagename.aspx") or, relative to the root of the web server on which the SCORM Engine is deployed (ex: "/ScormEngine/pagename.aspx").
Note: For simplicity, the file extension has been omitted from the page names included below. The file extension will either be ".aspx" for .NET deployments or ".jsp" for Java deployments.
Note: As of v2009.1 of the SCORM Engine, URLs that specify a directory have the option of including or not including the trailing slash ("/").

ScormEngineUrl - When doing a basic installation of the SCORM Engine, this is the only URL setting that must be included. It is simple a URL to the root location of the SCORM Engine on the web server. If this setting is used, the rest of the URL settings can be omitted.

Example: "/ScormEngine/ScormEngineInterface/"

ScormEngineFilesystemRoot - When using the optional offline/mobile SCORM Engine add-on you will need to define this location.

Example: "c:\inetpub\wwwroot\scormengine"

RemoteRequestProcessorUrl - URL to the "ProcessAiccRequest" file. This URL should be in the same domain that content is launched in. In a standard deployment, this setting should contain the same value as the "CentralAiccRequestProcesorUrl" setting, or it can be omitted. In a cross domain, central/remote deployment, this setting should point to the "ProcessAiccRequest" page in the SCORM Engine instance deployed to the content server. The URL should either be fully qualified, or relative to the location from which the content is lanched.

Example: "/ScormEngine/ScormEngineInterface/ProcessAiccRequest.aspx"

CentralAiccRequestProcessorUrl - URL to the "ProcessAiccRequest" file. This URL should be in the same domain that LMS resides in. In a standard deployment, this setting should contain the same value as the "RemoteAiccRequestProcesorUrl" setting. In a cross domain, central/remote deployment, this setting should point to the "ProcessAiccRequest" page in the SCORM Engine instance deployed to the LMS server and the URL should be fully qualified.

Example: "/ScormEngine/ScormEngineInterface/ProcessAiccRequest.aspx"

ScormResultProcessorUrl - URL to the "RecordResults" file. This URL should be in the same domain that the content resides in.

Example: "/ScormEngine/ScormEngineInterface/RecordResults.aspx"

CentralWebServiceUrl - URL to the "RuntimeDataExchange.asmx" file. This setting is only needed for cross domain, central/remote deployments. This URL should be an fully qualified URL to the "RuntimeDataExchange.asmx" file on LMS server. Currently this functionality is only available in the .NET implementation of the SCORM Engine.

Example: "http://www.lmsserver.com/ScormEngine/ScormEngineInterface/RuntimeDataExchange.asmx"

ImportWebServiceUrl - URL to the "ImportService.asmx" file. This setting only applies to deployments using the SCORM Engine's import controls on a server other than the LMS server. This URL should be an fully qualified URL to the "RuntimeDataExchange.asmx" file on LMS server. Currently this functionality is only available in the .NET implementation of the SCORM Engine.

Example: "http://www.lmsserver.com/ScormEngine/ScormEngineInterface/ImportService.asmx"

UrlToCentralLaunchPage - URL to the "defaultui/launch" page. This setting is only used by the Noddy LMS to determine how to redirect to the SCORM Engine when launching

content. In a production deployment where the Noddy LMS is not deployed, this setting is not required. When creating a custom skin of the SCORM Engine during integration with an LMS, it may be helpful to change this setting to point to the launch page in the directory containing your skin (vs the "defaultui" directory) in order to test the functionality of your skin.

Example: "/ScormEngine/ScormEngineInterface/defaultui/launch.aspx"

RemoteLaunchPageUrl - URL to the *directory* containing the "deliver" page. This directory can vary when using custom skins of the SCORM Engine. When using a cross domain, central/remote deployment, this setting should be a fully qualified URL pointing to the server from which the content will be served.

Example: "/ScormEngine/ScormEngineInterface/defaultui/"

ScormEngineScriptsUrl - URL to the *directory* containing the "launch" page. This directory can vary when using custom skins of the SCORM Engine. When using a cross domain, central/remote deployment, this setting should be a fully qualified URL pointing to the server from which the content will be served.

Example: "/ScormEngine/ScormEngineInterface/scripts"

RedirectOnExitUrl - URL to which the SCORM Engine should redirect the user after the user exits the SCORM Engine. This URL should point to a location in the host LMS.

Example: "/ScormEngine/NoddyLms/NoddyLms.aspx"

StylesheetUrl - URL to the stylesheet used by the current SCORM Engine skin.

Example: "/ScormEngine/ScormEngineInterface/defaultui/defaultstyles.css"

UrlToLaunchHistoryControlResources - URL to the "UploadImportControl" directory that must be deployed with the launch history report web control. This directory contains the resources the user interface of the launch history report needs to display properly. Only applicable to applications using the launch history report web control.

Example: "/ScormEngine/NoddyLms/UploadImportControl/"

EngineUtilsHelperUrl - URL to the "EngineUtilsHelper" aspx or jsp (depending on your platform) that is used for displaying, Properties Editor, the Registration Report, and other ajax-enabled functionalites. This URL is optional and is only necessary if your EngineUtilsHelper file is not in the folder represented by the "ScormEngineUrl" setting.

Example: "/ScormEngine/ScormEngineInterface/EngineUtilsHelper.aspx"

Upload Import Control

These settings are used by the web control that provides an interface for uploading content and importing it into the SCORM Engine. If this web control is not used by the LMS integrated with the SCORM Engine, then these settings are not required.

WebPathToContentroot - HTTP path to the directory in which uploaded courses should be stored. This directory should map to the directory specified in "FilePathToContentRoot".

Example: "/courses/"

FilePathToContentRoot - File path to the directory in which uploaded courses should be stored. This directory should contain the files served by the path specified in "WebPathToContentroot". Individual courses will be placed in subdirectories within this directory.

Example: "C:\inetpub\wwwroot\courses"

FilePathToUploadedZippedPackage - Courses are uploaded as zip files. This setting specifies a temporary directory that zipped courses are uploaded to prior to their extraction into the directory specified in "FilePathToContentRoot". After extraction, zip files are deleted from this directory.

Example: "C:\inetpub\wwwroot\courses\uploads"

UrlToUploadResources - URL to the "UploadImportControl" directory that must be deployed with the upload import web control. This directory contains resources the user interface of the upload control needs to function properly.
Example: "/ScormEngine/NoddyLms/UploadImportControl/"

UrlToLetsiRtwsEndpoint - URL to the LETSI RTWS endpoint which will be appended in an initial browser request for a LETSI RTWS enabled package as the LETSI_RTWS_URL. Only necessary if using the RTWS functionality.

Registration Instance and Package Versioning

These settings control how and when a new versions of packages and registrations are created. A version of a registration is called an "instance".
CreateRegistrationIfNeeded - When the SCORM Engine is launched with an external registration id that does not already exist, this setting controls whether a new registration is created for that id (setting="true") or if an error is thrown (setting="false"). This setting should be set to "false" only when SCORM Engine registrations are pre-created by the LMS via SCORM Engine API calls. Possible values: "true" or "false".
WhenToRestartRegistration - Controls the logic that is used to determine if a new instance of a registration should be created on launch.

Setting Value	Behavior
"1"	Never create new registration instances. Always relaunch the registration using the existing set of tracking data.
"2"	Create a new registration instance if there is a newer version of the package being delivered and the current registration instance is completed.
"3"	Create a new registration instance whenever there is a newer version of the package being delivered.
"4"	Create a new registration instance whenever the user launches a registration that has previously been completed.
"5"	Create a new registration instance whenever the user launches a registration that has previously been satisfied.
"6"	Create a new registration instance if there is a newer version of the package being delivered and the current registration instance is satisfied.

IsPackageVersioningEnabled -When the SCORM Engine's import routines are called with an external package id that already exists, this setting controls whether a new package version is created or whether an error is thrown. Possible values: "true" (create new package versions) or "false" (throw an error).

SystemHomepageUrl - Get the absolute URL of the cannonical, permanant, homepage for this system. Ideally this really is the homepage a user would use to access the system, but

this must be cannonical and permanant, that is: it is a single URL the system can be identified by. Used by Tin Can when creating Actors from users in the system.

Optional SCORM Engine Features

These settings control the behavior of some optional SCORM Engine features that might not apply to all installations.
2004Enabled - Tells the SCORM Engine whether this installation has the capability to deliver SCORM 2004 content. This setting does not actually affect the SCORM Engine's ability to deliver SCORM 2004 content, instead it just tells the SCORM Engine whether or not to issue a warning message when the user attempts to import SCORM 2004 content. Possible values: "true" (SCORM 2004 support is enabled) or "false" (SCORM 2004 support is not enabled).
SSPEnabled- Tells the SCORM Engine whether this installation has the capability to deliver SSP content. This setting does not actually affect the SCORM Engine's ability to deliver SSP content, instead it just tells the SCORM Engine whether or not to issue a warning message when the user attempts to import SSP content. Possible values: "true" (SSP support is enabled) or "false" (SSP support is not enabled).
SSPSizeAllocation - When using SSP, this setting determines the maximum amount of storage that a given course can request for a given registration. This setting is an integer that specifies a number of bytes.
Example: "1048576" (corresponds to 1 MB of storage)
UseCompressedJavascript - The SCORM Engine sends a lot of JavaScript code to the user's browser to implement all of the required SCORM functionality. To speed up the loading process, by default, this code is compressed and consolidated. This setting controls whether the compressed version of the code is delivered to the browser (the best setting for production environments) or whether the raw, uncompressed code is delivered to the browser (useful for development and debugging). Possible values: "true" (deliver compressed code) or "false" (deliver raw code)
AiccSessionIdExternalConfigExclusions - If the AICC Url, with SID included, is too long, this parameter can be used to exclude non-essential external configuration parameters from serialization.
AiccUseLegacySidFormatForExistingRegs - The SCORM Engine is now using GUIDs for the AICC session ID (backed by the ScormAiccSession table). However, for continuity, we need existing registrations to continue to use the long tilde-delimited format. This parameter provides the ability to override that logic and force even existing registrations to use the GUID format.
EnableExternalIdEncryption - Value that determines if external IDs should be encrypted by default.
IntegrationEncryptionPassword - String used to generate the encryption key for securing URLs passed between the web services integration and the SCORM Engine. This should be set to a long random string.
SMTP_Host - Host name of the SMTP server used for email functionality. Used by the PENS system.
SMTP_Port - Port number of the SMTP server used for email functionality. Used by the PENS system.
SMTP_User - User name used for authentication of the SMTP server used for email functionality. Used by the PENS system.
SMTP_Password - User password used for authentication of the SMTP server used for email functionality. Used by the PENS system.

SMTP_UseSecureConnection - Whether to use a secure connection when communicating with the SMTP server used for email functionality. Used by the PENS system.
Pens_Mail_Receipt_From - The address that the PENS email receipts will be from.
Pens_Mail_Alert_From - The address that the PENS email alerts will be from.
Pens_Process_Sync - Whether PENS request should be processed synchronously (recommended for troubleshooting only).
Pens_ByPass_SSL_Validation - If set to true, PENS will ignore SSL certificate validation failures. Recommended for testing purposes only.
RtwsSessionTimeoutHours - Timeout of LETSI RTWS session in hours. RTWS servers may leave sessions enables indefinitely and should leave them enabled for at least 24 hours.
TinCanRootAccount - This is a colon-delimited name and password like "admin:mypass" which can be used to authenticate against the TCAPI with administrator rights. This parameter is required to use the built-in console's Tin Can statement viewer.
ConsolePassword - Password to gain access to the /tools/console/console.aspx page which contains some SCORM Engine diagnostic tools and hooks to the Tin Can Statement viewer.

Debug Settings

These settings control the amount of debugging information that is recorded by the SCORM Engine. There isn't much of a performance penalty for recording this information, so we recommend that these settings typically be left at their default values to assist with troubleshooting. In this context, "audit" means recording basic debug information about what happend and when. "Detailed" means recording the precise details of how each action was executed. In order for the "detailed" information to be properly recorded, the "audit" level information must also be captured.

KeepAuditLog - Determines whether server-side debug information is captured at the audit level. This log tracks which server-side pages where requested and when. Possible values: "true" (record information) or "false" (don't record information).

KeepDetailLog - Determines whether server-side debug information is captured at the detailed level. This log tracks the execution of server-side pages. Possible values: "true" (record information) or "false" (don't record information).

KeepSoapLog - When used a cross domain, central/remote architecture, this setting determines if the exact contents of SOAP web services calls between the central and remote instances are logged. Possible values: "true" (record information") or "false" (don't record information).

DebugControlAudit - Determines whether client-side information about the overall execution of the SCORM Engine is recorded at the audit level. "Control" information tracks what was launched when as well as the communication with the server. Possible values: "true" (record information) or "false" (don't record information).

DebugControlDetailed - Determines whether client-side information about the overall execution of the SCORM Engine is recorded at the detailed level. Possible values: "true" (record information) or "false" (don't record information).

DebugRteAudit - Determines whether SCORM runtime calls from SCOs are logged are recorded to the client-side debug log at the audit level. Possible values: "true" (record information) or "false" (don't record information).

DebugRteDetailed- Determines whether SCORM runtime calls from SCOs are logged are recorded to the client-side debug log at the detailed level. Possible values: "true" (record information) or "false" (don't record information).

DebugSequencingAudit - Determines whether the execution of the SCORM sequencing logic is recorded to the client-side debug log as the audit level. Possible values: "true" (record information) or "false" (don't record information).

DebugSequencingDetailed - Determines whether the execution of the SCORM sequencing logic is recorded to the client-side debug log as the detailed level. Possible values: "true" (record information) or "false" (don't record information).

DebugSequencingSimple - Determines whether the execution of the SCORM sequencing logic is recorded to the client-side debug log in the "simple" format when available. Possible values: "true" (record information) or "false" (don't record information). When true and enabled, this setting will disable DebugSequencingAudit and DebugSequencingDetailed.

DebugLookAheadAudit - The SCORM Engine executes "look ahead" runs of the SCORM sequencer whenever pertinent data is changed in order to determine whether or not to enable/disable/show/hide the various navigational controls available to the user. This setting determines if these executions are recorded to the client-side debug log at the audit level. Possible values: "true" (record information) or "false" (don't record information).

DebugLookAheadDetailed - Determines whether the execution of the look ahead SCORM sequencing is recorded to the client-side debug log as the detailed level. Possible values: "true" (record information) or "false" (don't record information).

DebugIncludeTimestamps -Determines whether or not the client-side debug logs should include time stamps indicating when audit-level events occur. Possible values: "true" (record time stamps) or "false" (don't record time stamps).

Logging - Both the .Net and Java implementations of the SCORM Engine include the capability to integrate with a server-side logging framework. The SCORM Engine uses Apache's log4net and log4j to store rolling logs of server-side activity on the file system. These logging systems have many settings that are stored in the web.config file in .Net and the log4j.properties file in Java. Refer to the appropriate logging system's website for information on configuring these systems. (Note that to use the log4net system, the "NETWORK SERVICE" user will need to have read/write permissions to the logging directory.)

Central / Remote Architecture

These settings apply to the use of the cross domain, central/remote architecture.
UseCrossDomainWebServices - Determines whether or not the cross domain, central/remote architecture is in use. If this setting is set to "true", requests to persist data will be forwarded to the location specified in the URLs specified in the CentralAiccRequestProcessorUrl and CentralWebServiceUrl settings. If this setting is set to "false", requests will be directly processed. Possible values: "true" or "false".
WebServiceRetries - If using web services, this setting determines the maximum number of times the remote instance will attempt to contact the central instance in the event of an error. Once the maximum number of retries has been reached, the remote instance will assume that communiation with the central instance has been lost and notify the user that an error has occured. This value is specified as an integer.

Example: "3"

WebServiceRetryInterval - If the remote instance needs to retry its communication with the central instance, this setting determines how long the remote instance will wait before resending the request. This value is specified in milliseconds. When using a central/remote architecture, the maximum time that could be spent retrying requests (calculated as WebServicesRetries * WebServiceRetryInterval) should be significnatly less than the default CommCommitFrequecy package property to prevent the remote server from being overloaded in the event of a failure of the central server. This maximum time value also needs to be less than the ASP.NET / JSP page timeout value.

Example: "5000" (corresponds to 5 seconds)

UseImportWebServices - Determines whether or not import controls should use web services to invoke import on a central server. Possible values: "true" or "false"

Updating Your SCORM Engine for .Net

SCORM Engine 2010.1 and higher

Between major releases of the SCORM Engine we may make point releases that fix bugs and add small pieces of functionality that are needed by our clients. When you update your SCORM Engine implementation to one of these point releases we recommend that you follow these instructions so that we can continue to support you easily.

If you have any questions about these instructions, or you think they are not optimal for your deployment scenario, then please contact us and we will help you through the upgrade process.

The SCORM Engine interface is a web application that is customized for our clients by adding a custom Client Integration dll to its bin folder and configuring its web config settings through the SCORMEngineSettings.config file. We test our configurations with all files at the same code level so we encourage you to think of the files in the SCORM Engine directory as a single unit, despite the fact we may be providing a patch that only affects a handful of JavaScript files. Some updates may include database scripts for schema or stored procedure changes but we will explicitly call out if/when that is necessary. The standard upgrade instructions depend on your deployment scenario:

Single SCORM Engine Web Application, default user interface

In the most common deployment scenario where the SCORM Engine is deployed separately to the Client LMS as a single web application, and the Client LMS uses the standard UI files located in <ScormEngineInterfaceDir>/defaultui. You'll essentially need to completely replace the SCORM Engine application while keeping your own integration DLL and configuration file. Follow these steps (assuming the SCORMEngineInterface web app is located at <ScormEngineInterfaceDir>):

Copy away the client integration dll:
<ScormEngineInterfaceDir>/bin/<ClientIntegration>.dll
Copy away the client settings file:

<ScormEngineInterfaceDir>/SCORMEngineSettings.config
Delete <ScormEngineInterfaceDir>.
Unpack the ScormEngineInterface web app to <ScormEngineInterfaceDir>.
Copy back the client integration dll to <ScormEngineInterfaceDir>/bin/.
Copy back the SCORMEngineSettings.config to <ScormEngineInterfaceDir>.

Single SCORM Engine Web Application, custom user interface.

In this deployment the Client LMS has its own set of SCORM Engine UI files, usually based on the /defaultui files, in a separate web application. It is rare that we will have made changes to the UI files in a point release. If we have then we will indicate the changes so that the client can merge these changes into their modified ui. Additionally:

Replace ScormEngineInterface as in the "Single SCORM Engine Web Application, defaultui" steps.
Copy <ScormEngineInterface>/bin/RusticiSoftware.ScormEngine.dll to the bin directory of the webapp hosting the ui.

Single Central SCORM Engine, multiple Remote SCORM Engines

In this deployment there is a single SCORM Engine co-located with the database and one or more remote SCORM Engines co-located with the course content.

Replace Central ScormEngineInterface as in the "Single SCORM Engine Web Application, defaultui" steps.
Repeat this process for each Remote SCORM Engine.

Upgrading the SCORM Engine to v2013.1 from v2012.2

Step 1: Update the application files

This update can be performed using the standard upgrade process for point releases, found in the Updating your Scorm Engine for .Net document.

Step 2: Update your database

In the SCORM Engine update files that were delivered to you, open the "db" folder and then open the folder corresponding to the database you are using. Then open the folder

"2013.1" and then the "upgrade" folder". In here, you will find a SQL script file that will upgrade the SCORM Engine database objects to be compatible with v2013.1. Simply run execute this script against the database containing the SCORM Engine tables to complete the upgrade. We recommend backing up your existing database before attempting the upgrade to protect against the unlikely event of an error during script execution.

Step 3: Custom UI Updates

If you using a custom ui, meaning something other than scormengine/defaultui/deliver.aspx[or jsp], you will need to make the following additions to to take advantage of the new IE Compatibility Mode package property. If this step is skipped, your player will continue to function but it will continue to use the hard-coded META tag, if any.

Remove code in RED. Add code in GREEN.

***Deliver.aspx or jsp**

```
<html>
        <head>
        <meta http-equiv="X-UA-Compatible" content="IE=EmulateIE8" />
<%=Data.InternetExplorerCompatibilityMetaTag %>
```

***Intermediate.aspx or jsp**

```
<html>
        <head>
<%=Data.InternetExplorerCompatibilityMetaTag %>
```

Step 4: Upgrade historical Tin Can Statements

If you have existing Tin Can data that you would like to be available in your 2013.1 installation, we've provided a utility to migrate the data into your new installation. With your SCORM Engine upgrade, you should have also received a copy of TinCanSchemaConverter appropriate for your platform (.NET or Java).

To run the converter:

Extract the contents on a workstation or server that has access to your SCORM Engine database.

Go to the extracted directory, and open SCORMEngineSettings.config (or SCORMEngineSettings.properties for Java installations)

Edit the values of DataPersistanceEngine and DatabaseConnectionString to the correct values for your SCORM Engine database.

Run statement-converter.bat (or statement-converter.sh). If an error occurs, please contact us and we'll assist you to resolve the issue.

When the converter is finished, confirm the historical data has been upgraded by viewing or fetching your Tin Can statements using the Tin Can Statement Viewer in the SCORM Engine console.