

SCORM Engine 2008.1

Documentation

Last Modified: September 21, 2009

[SCORM Engine Integration](#)

[Welcome](#)

[Working Together](#)

[The Kickoff Meeting](#)

[The Setup Phase](#)

[The Integration Phase](#)

[Import](#)

[Launch](#)

[Rollup and Reporting](#)

[Further Integration Considerations](#)

[Testing Phase](#)

[Going Forth](#)

[Material Completion](#)

[Certification](#)

[Support Process](#)

[Troubleshooting](#)

[Updates and Patches](#)

[Synchronized Code Bases](#)

[Using The Noddy LMS](#)

[History](#)

[What is the Noddy LMS?](#)

[Importing Courses](#)

[Import from an Uploaded .zip file](#)

[Import from the Filesystem](#)

[Import Single Course from URL](#)

[Create Package from Scratch](#)

[Web Controls](#)

[Importing Multiple Courses](#)

[Parser Warnings](#)

[Main Screen](#)

[New Registration](#)

[Update Package](#)

[Properties](#)

[Delete](#)

[Working With Versions](#)

[Preview](#)

[Properties](#)

[Delete](#)

[Registrations](#)

[Registration ID Fields](#)

[Status](#)

[Launch](#)

[Report](#)

[Reset](#)

[Delete](#)

[Clearing Data](#)

[Clear All State Data](#)

[Clear All Data](#)

[Instructions for Deploying the Noddy LMS for .Net](#)

[Prerequisites](#)

[Summary](#)

[Step 1: Copy the files to the file system.](#)

[Step 2: Configure IIS](#)

[Step 3: Configure the database](#)

[Step 4: Configure the application](#)

[Database Connection String](#)

[File Paths](#)

[Applying Changes](#)

[Step 5: Permissions](#)

[Step 6: Launch](#)

[Step 7: Clean up \(optional\)](#)

[Instructions for Deploying the Noddy LMS for Java](#)

[Prerequisites](#)

[Summary](#)

[Step 1: Extract the files](#)

[Step 2: Configure the database](#)

[Step 3: Deploy the *.war files](#)

[Step 4: Connect the web server with the new Java applications](#)

[Step 5: Create a JNDI Data Source](#)

[Step 6: Deploy the ScormEngineSettings.properties file](#)

[Step 7: Edit the settings in the SCORMEngineSettings.properties file](#)

[Step 8: Restart and launch](#)

[SCORM Engine Settings](#)

[Working with the SCORM Engine Settings](#)

[.NET](#)

[Java](#)

[The Settings](#)

[Integration Class](#)

[Data Persistence](#)

[Advanced Data Persistence Settings](#)

[URLs](#)

[Upload Import Control](#)

[Registration Instance and Package Versioning](#)

[Optional SCORM Engine Features](#)

[Debug Settings](#)

[Central / Remote Architecture](#)

[SCORM Engine Launch Parameters](#)

[Configuration](#)

[Registration](#)

[Package](#)

[ManifestDirPath and WebPath](#)

[Tracking](#)

[ForceReview](#)

[RegForCredit](#)

[CC](#)

[StartSCO](#)

[Serializing and Encoding](#)

[Common Configurations](#)

[Launch a registration "normally"](#)

[Launch a completed registration in review mode with no changes to the tracking data](#)

[Launch an imported course in preview mode with no tracking](#)

[Launch a course that does not "count" for credit, but should still be tracked](#)

[Launch a course directly from a manifest that has not yet been imported](#)

[Mode and Credit](#)

[SCORM Engine 2008.1 Upgrade Notes](#)

[Significant Integration Changes](#)

[The "central" and "remote" web applications have been consolidated.](#)

[The SCORM Engine libraries have been consolidated into a single library.](#)

[A new default UI has been created.](#)

[A formalized public API was added.](#)

[A ScormEngineManager class has been added to formalize the public API into the SCORM Engine that LMS integrations should be using.](#)

[Upgrading from SCORM Engine 2007.1](#)

[Step 1: Change your library references](#)

[Step 2: Utilize the new SCORM Engine web application](#)

[Step 3: Upgrade your Database](#)

[Step 4: Configuration Changes](#)

[Step 5: Integration Class Changes](#)

[Step 6: Web Controls References Change](#)

[SCORM Engine Course Properties](#)

[SCORM Engine Scalability](#)

[Introduction](#)

[Why is this such a hard question?](#)

[Deployment Variability](#)

[Integration Variability](#)

[Course Variability](#)

[Usage Variability](#)

[Empirical Evidence](#)

[Stress Testing Results](#)

[Methodology](#)

[Results](#)

[Conclusions](#)

SCORM Engine Integration

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 the 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 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

1. Introductions made and contact information exchanged
2. LMS tour
3. Unique identifier for package established
4. Unique identifier for registration established
5. Code and database platform(s) established
6. 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 an application called the "Noddy LMS". The Noddy LMS is nothing more than a simple interface to the SCORM Engine that allows you to import and launch courses. 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 Noddy LMS provides us a way to deploy and test the integration code before it is tied into your 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 and Noddy LMS, 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:

1. **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.
2. **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.
3. **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 compatibility. 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 determining 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 identifier 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.

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.

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 black 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.

For deeper problems that affect the operation of the SCORM Engine itself, we have a detailed server-side log that can be accessed. See [server-side debugging](#) for more information.

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.

Using The Noddy LMS

This document provides an overview of the Noddy LMS that ships with the SCORM Engine. It describes the functionality and purpose of the Noddy LMS and is intended to help you to understand how to use Noddy during your integration. For instructions on installing the Noddy LMS, please see [Instructions for Deploying the Noddy LMS for .Net](#) and [Instructions for Deploying the Noddy LMS for Java](#).

History

One of our top developers hails from the UK and he endures some good-natured ribbing from the rest of us when we can't understand his slang. One day he came to us and said:

Testing the SCORM Engine during integration is a right bit dodgy. How about we knock up a bespoke system that provides an interface for any client integration? It's easy peasy, all we need is to examine the external identifiers using reflection, knock up a quick UI and Bob's your uncle. No more than a fortnight for the full monty. Fancy that? What he meant is that our old Sample LMS wasn't compatible with custom integrations of the SCORM Engine. That incompatibility left us without a user interface for testing the SCORM Engine during most of the integration process. He had a really innovative idea for filling that void and further simplifying the integration process so we gave him the go ahead.

A little while later, he came back with the "Noddy LMS".

nod·dy - adj. (British slang) - used to describe something simple, small, or childish. We didn't quite understand what he was saying at first, but the name was kind of fun and actually described the product quite well. So here you have it, the "Noddy LMS". Please understand, there is nothing *naughty* about the *Noddy* LMS.

What is the Noddy LMS?

The Noddy LMS is a testing interface around the SCORM Engine. It exposes all of the basic SCORM Engine functionality through a simple graphical user interface. The SCORM Engine is a component that is meant to be used as part of a larger system, it can't stand alone. The Noddy LMS provides a simple framework in which the SCORM Engine can run. It provides

the most basic LMS functionality, the ability to import, deliver and track courses.

A key feature of the Noddy LMS is that it can interface with the SCORM Engine using any client integration layer (see [SCORM Engine Integration Architecture](#) for more information about client integrations). This abstraction is enormously useful during the [process for integrating the SCORM Engine](#) with an LMS. It allows us a constant interface to manipulate the SCORM Engine before, during and after the integration. Without the Noddy LMS, there would be a sample interface to use before the integration begins and you could use the integrated LMS once the integration is fully complete, but in the interim, there would be no easy and reliable way of accessing the SCORM Engine's functionality.

You can use the Noddy LMS for many things, including:

- Verifying correct installation of the SCORM Engine.
- Seeing how the SCORM Engine operates.
- Testing content in the SCORM Engine (or, use our free online [SCORM Test Track](#) tool).
- Test the integration with an LMS as it proceeds.

The Noddy LMS is particularly useful during the phase of LMS integration during which the import mechanism is being created and to visualize the creation of new package "versions" and new registration "instances".

Importing Courses

The first thing you will want to do in the Noddy LMS is import some content to work with. Clicking the "Import" link from the top menu bar will bring you to the screen below.

The screenshot shows the 'Import' page of the Noddy LMS. The page is titled 'The Scorm Engine Noddy LMS (.NET) - Mozilla Firefox'. The browser address bar shows the URL 'http://localhost/ScormEngine/NoddyLms/Import.aspx?showhome=0&configuration=...'. The page header includes 'RUSTICI SOFTWARE' and 'NODDY LMS'. Below the header is a 'HOME' link. The main content area contains four sections for importing courses:

- Import from an Uploaded .zip file**: A form with a file input field, a 'Browse' button, and an 'Import' button. A '[Show Help]' link is also present.
- Import from the Filesystem**: A form with a 'Virtual Web Path' input field (example: 'c:\inetpub\wwwroot\') and an 'Import' button. A '[Show Help]' link is also present.
- Import Single Course from URL**: A form with a 'Manifest URL' input field (example: 'http://www.mycompany.com/content/courses/course1/manifest.xml') and an 'Import' button. A '[Show Help]' link is also present.
- Create Package from Scratch**: A form with 'Title' and 'URL' input fields, a 'Type' dropdown menu set to 'SCORM 1.1', and an 'Import' button.

The footer of the page contains '© 2008 Rustici Software™ East'.

This import page provides four options for getting content into the Noddy LMS.

Import from an Uploaded .zip file

Most often, SCORM courses are delivered in a zip file (the technical SCORM term for this is a "PIF" or "package interchange file"). This control allows you to simply click the "Browse" button to select a zip file to be uploaded. This is the easiest and most common method of importing a course.

Import from the Filesystem

There are times where it isn't practical to upload a zip file directly. For instance that zip could be unusually large, the course could already be extracted on the web server, or there could be an advanced content delivery scheme that requires manual deployment. For these cases, this option allows you to simply enter a path to the course to be imported. The path is entered as a web path that maps to a file path on this server. For instance, if your course resides in "c:\inetpub\wwwroot\courses\my_course" and the web root maps to "c:\inetpub\wwwroot", you would enter "/courses/my_course/" in this text box.

Import Single Course from URL

Similar to the "Import from the Filesystem" this option allows you to import a course that has already been extracted and deployed to a web server. The difference between the two options is that this option will retrieve the course via HTTP from any server vs the other option which opens the course directly from the file system of the hosting server. Because of this distinction, the full path to the course's descriptor file must be specified (for SCORM, the "imsmanifest.xml" file, for AICC a file with the ".au" extension). For example "http://www.mycontentserver.com/courses/my_course/imsmanifest.xml".

Create Package from Scratch

The other import mechanisms all rely on the course descriptor files to provide information about the course that is required for import. In rare cases, courses don't have descriptor files (especially AICC courses). This option allows you to create a new course by manually entering the minimum information needed for a course. Simply enter a title for the course, the URL to the launch page for a course (can be fully qualified or relative to the root of the current server) and select the learning standard under which the course communicates to the LMS.

Web Controls

The user interfaces for these four options are all available as reusable web controls that you

can use directly within your application.

Importing Multiple Courses

The first two options have a handy feature that allows you to import many courses at once. If you upload a "zip of zips" to the first option, it will treat each zip file contained within the top level zip file as a separate course and import each of them individually. Similarly, if a directory is passed to the second option (vs specifying the full path to the descriptor file), it will iterate through the contents of that directory (and its sub-directories) looking for courses to import.

Parser Warnings

Not all SCORM courses are perfect. Often content developers make minor mistakes in the structure of the content or in the details of their descriptor files. The SCORM Engine can detect many of these common mistakes and work around them. Whenever the SCORM Engine detects a problem that it knows how to deal with, it will issue an informative parser warning. These warnings usually do not affect the operation of the content, but they are worth noting should any odd behavior show up down the road. Also, it's worth trying to fix any and all parser warnings in the content if the content is intended to run in other LMS's.



Main Screen

Once courses have been imported, they will be listed on the home page with the most recently imported course on top. For each course, some high level information is displayed:



On the left is the external package id. The external package id is the value that the system

with which the SCORM Engine is integrated uses to identify a course in the SCORM Engine.

Integration Note: In the initial installation of the Noddy LMS, this field is labeled "CourseId". If the SCORM Engine is integrated with another LMS, the data and labels in this box will change to correspond with the external package id fields specific to that integration.

Next is the course title. This title was parsed from the course descriptor files during the import process.

On the right side, there are two boxes with numbers in them. The number on the left indicates the number of versions of this course that have been imported. This document will discuss course versioning in a later section. The other number indicates the number of registrations that have been made to deliver this course. A registration is an instance of a user taking a course. Creating registrations will be discussed in the next section.

Beneath each course title are links for performing operations on that course.



New Registration

The "New Registration" link allows you to (of all things) create and launch a new registration. Clicking on the link will bring up a dialog box prompting you to enter information to be used to identify the new registration.

CourseId 1001 **Photoshop Example -- Linear** 1 version 0 registrations

New Registration Update Package Properties Delete

Create a new registration by filling in the values and clicking launch below. [X]

UserName

CourseId

Launch URL
/ScormEngine/ScormEngineInterface/defaultui/launch.aspx?configuration=®istration=CourseId|1001

Launch

In the initial installation of the Noddy LMS, only a "UserName" is required to be entered. This field can contain any arbitrary text, it just has to uniquely identify the registration within this package. Simply enter a value in the "UserName" field and click "Launch".

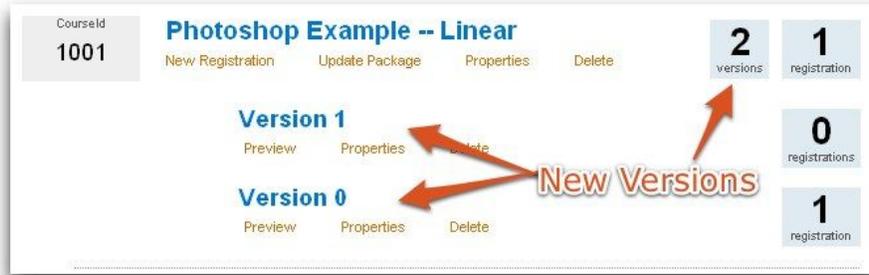
Integration Note: The list of fields that will be displayed will vary if a different integration is used with the Noddy LMS. This list represents all of the components of the external registration id for the current integration used by the SCORM Engine. Fields that are grayed out represent the external package id (which is often part of the external registration id) and do not need to be filled in because they are inferred from the package this registration is associated with. **Integration Note:** As you enter the fields for the external registration id, the "Launch URL" text will change. This URL represents the URL your system would redirect to in order to launch a course under this registration id. This feature can be handy when coding the launch portion of your integration. Simply copy and paste the link into your code to ensure that the formatting is correct.

Also, if your integration requires the use of external configuration information, the Noddy LMS can be configured to append this configuration information to the launch string. Simply add a querystring parameter "configuration=XXX" to the launch URL of the Noddy LMS.

Update Package

The "Update Package" link brings you back to the import screen. Using the standard import controls, you can select a new course file to update the existing course. This action will create a new version of the course. By default, existing registrations of the course will remain on the original version of the course and new registrations will get the new version of the course.

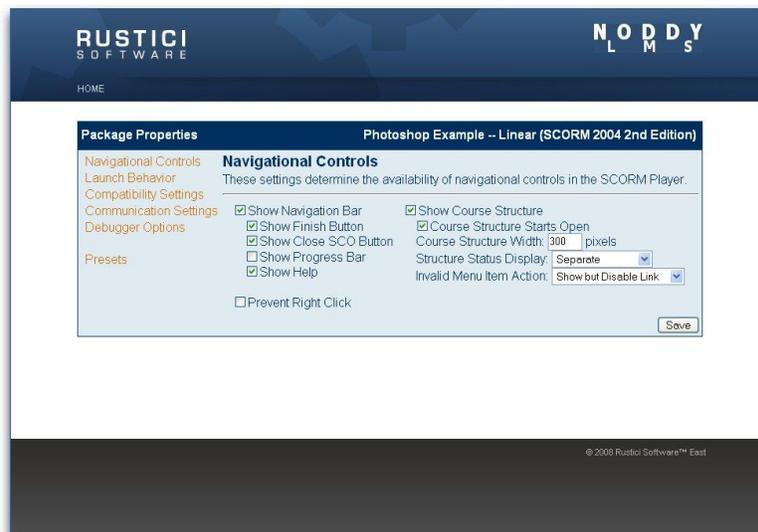
Integration Note: When a registration is launched and new versions of the course are available the SCORM Engine might decide to create a fresh registration for the latest version of the course. The circumstances under which this happens can be customized in your integration layer. Once the new version has been created, the version count will increase and clicking on the course title will display information about each of the versions independently.



A later section will discuss the options for working with individual package versions.

Properties

The "Properties" link takes you to the "Package Properties" page. This page allows you to edit the properties that control how a course is delivered in the SCORM Engine. See the [SCORM Engine Package Properties](#) document for more information on the individual properties. Changing properties from this link will change the properties for all versions of the course.



Delete

The "Delete" link simply deletes all versions of this package and any registrations associated with those packages.

Working With Versions

Clicking on a course title will bring up a list of each version of the course that has been imported. Each version has individual operations that can be performed on it.



Preview

It is often helpful to launch a course to observe its behavior without the overhead and tracking of a registration. The "Preview" link simply launches a course without any tracking.

Properties

The "Properties" link on a course version brings you to the "Package Properties" page just like the "Properties" link under the course title. The difference is that this "Properties" link edits package properties for this version only. The link under the course title edits properties for all versions simultaneously.

Delete

The "Delete" link simply deletes this version of the package and all of the registrations associated with this version.

Registrations

Clicking on a version name will bring up a list of the registrations associated with that version of the package. The status boxes to the right of each package and version indicate how many registrations are associated with the package/version. The registration box contains information about the registration id, the registration status and links to operations that can be performed on the registration.

Registration ID Fields

The "Registration ID Fields" at the top of the registration box list all of the external registration id fields that identify this registration. These fields are the same fields displayed when creating a registration except for the addition of the "InstanceId" field. An "instance" is a version of a registration. By default in the SCORM Engine, whenever a registration is launched, the data is always tracked under the same instance id. However, there are times when for a given registration, multiple instances may appear (because of package versioning or perhaps business rules governing the retaking of courses).

Status

The "Status Information" provides a high level summary of the current status data for the course. All of these values are determined from the data the course has reported to the SCORM Engine and the sequencing rollup rules the course author specified. (More technically, this data is the activity state data for the root activity in the content aggregation.)

- "Complete" indicates whether or not the learner has made it through all of the course material.
- "Success" indicates whether or not the learner has mastered the course material. Success status is akin to passed or failed.
- "Score" reflects the level of mastery, akin to a test score. Value is displayed on a scale of -100 to 100 (normal results will be 0-100, but negative values are valid in SCORM, they indicate that not only did you not master the material, but you actually did something much worse)
- "Total Time" indicates the amount of time (in seconds) that the learner has spent in the course.

The first three elements can all have a value of "unknown" This state indicates that the content has not yet reported any data.

Launch

The "Launch" link will re-launch this registration. Any data from previous attempts will be maintained on subsequent launches.

Report

Clicking on the "Report" link will bring up the "Activity Report" page. This page lists detailed information about the current registration. All of the available SCORM data is displayed on this page.

The screenshot displays two main sections: 'Instance Information' and 'Activity Report'. The 'Instance Information' section is divided into two sub-sections: 'Registration Fields' and 'Scoring Information'. The 'Registration Fields' section shows 'UserName: joeuser', 'CourseId: 1001', and 'Instancelid: 0'. The 'Scoring Information' section shows 'Complete: unknown', 'Success: unknown', 'Score: unknown', and 'Total Time: 8'. The 'Activity Report' section is a tree view showing a hierarchy of activities. The root is 'Photoshop Example -- Linear [AGGREGATION]', which is expanded to show 'Satisfied: False', 'Completed: False', 'Progress Status: True', 'Attempts: 1', and 'Suspended: False'. Underneath are 'Activity Objective #1', 'Introduction [ASSET intro.htm]', and another 'Activity Objective #1'. The 'Introduction' item is expanded to show 'Satisfied: True', 'Completed: True', 'Progress Status: True', 'Attempts: 1', and 'Suspended: False'. Below this is another 'Activity Objective #1', which is expanded to show 'Runtime Data'. At the bottom of the tree are 'Module 1 -- Basics [AGGREGATION]', 'Module 2 -- Enhancing Images [AGGREGATION]', 'Module 3 -- Blending Images [AGGREGATION]', and 'Exam [AGGREGATION]'.

Reset

The "Reset" link will clear out all of the tracking data associated with this registration and allow you to start fresh.

Delete

The "Delete" link will completely delete this registration from the database.

Clearing Data

The menu bar contains two links next to the "Import" link that are useful for clearing out the data in the Noddy LMS. Be careful when using these and other delete functions, especially if the Noddy LMS is running against an instance of the SCORM Engine that is being integrated with your LMS. In that case, you are operating on real data in the SCORM Engine.



Clear All State Data

The "Clear All State Data" link will recreate all registrations, resetting their state, while leaving the packages and their versions intact.

Clear All Data

The "Clear All Data" link completely resets the SCORM Engine database. It deletes all packages and registrations.

Instructions for Deploying the Noddy LMS for .Net

This document will describe how to install the .Net version of the SCORM Engine along with the Noddy LMS on a Windows machine running SQL Server. It will describe a basic installation designed to get the SCORM Engine and Noddy LMS up and running quickly. These instructions were derived from an installation on Windows XP running SQL Server 2005.

Prerequisites

This document assumes that the following are installed and properly configured on the target computer:

7. [.Net Framework 2.0+](#)
8. [Internet Information Services \(IIS\)](#)
9. [ASP.NET](#)
10. [SQL Server 2000+](#)

Summary

Here's the quick version of what needs to be done to install the SCORM Engine and Noddy LMS. These steps are outlined in detail below.

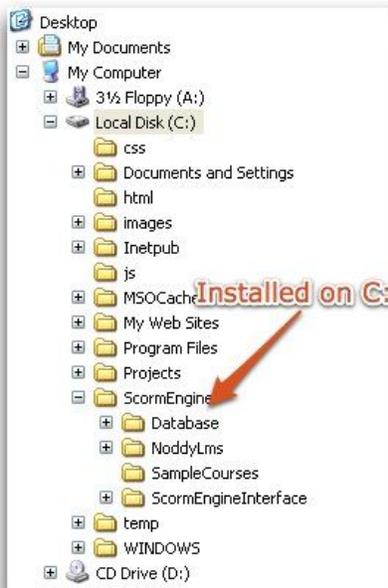
1. Extract the files in the deployable to your install location
2. Create a virtual directory in IIS that points to the location of the files (it should be

called ScormEngine and be at the root of the web server to save some configuration effort)

3. In IIS, make the "ScormEngine", "ScormEngine/NoddyLms" and "ScormEngine/ScormEngineInterface" directories applications. Set them to run under ASP.Net 2.0 or higher
4. Create a new database. Run the first two scripts (in order) listed in the database folder of the deliverable to create the SCORM Engine's database objects.
5. Open the "ScormEngineInterface/ScormEngineSettings.config" file. Change the "DatabaseConnectionString" setting to point to the database from step 4. Change the "FilePathToContentRoot" and "FilePathToUploadedZippedPackage" settings to point to the installed location of the ScormEngine files. You will then need to restart IIS.
6. Make sure that the "IUSR_ComputerName", "ASPNET" and "NETWORK SERVICE" users have Modify permissions to the ScormEngine directory and everything underneath it.
7. Launch the Noddy LMS at http://SERVER_NAME/ScormEngine/NoddyLms/noddylms.aspx.

Step 1: Copy the files to the file system.

The delivery you have should be a zip file containing a single directory called "ScormEngine" in addition to this document. Extract the zip file and then copy the "ScormEngine" directory to wherever you would like to install the SCORM Engine. For instance, move it to the "c:" drive to create "c:\ScormEngine" or move it to the wwwroot directory to create "c:\inetpub\wwwroot\ScormEngine". We do not recommend putting the SCORM Engine in any user specific directories like the desktop or My Documents as these locations can lead to difficulties with permissions. On some versions of Windows, the Program Files directory can also present permissions problems, but the last step of these instructions will describe the permissions the Scorm Engine requires to operate.



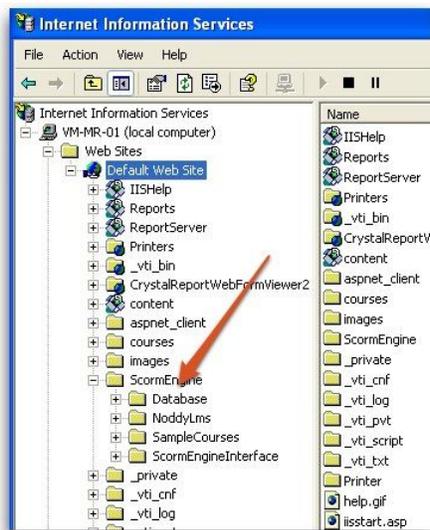
Step 2: Configure IIS

The next step is to configure IIS to serve the SCORM Engine application.

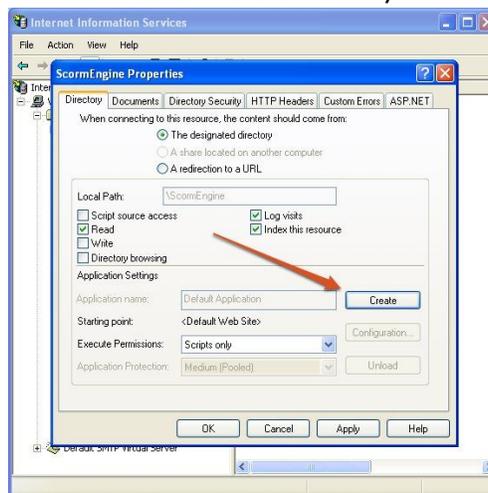
Open Internet Information Services (IIS). It is under Control Panel -> Administrative Tools.

What to do next depends on where you copied the files to in Step 1.

If you put the files in the wwwroot directory, then under the Default Web Site you will see a directory called ScormEngine.

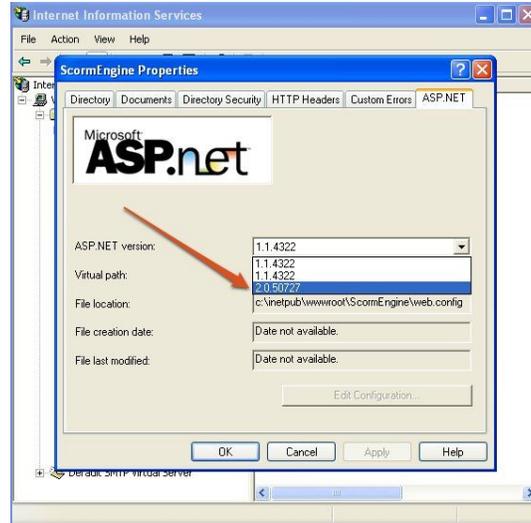


Right click on this directory, select Properties and then press the "Create" button in the "Application Settings" section. This will turn the directory into a web application.

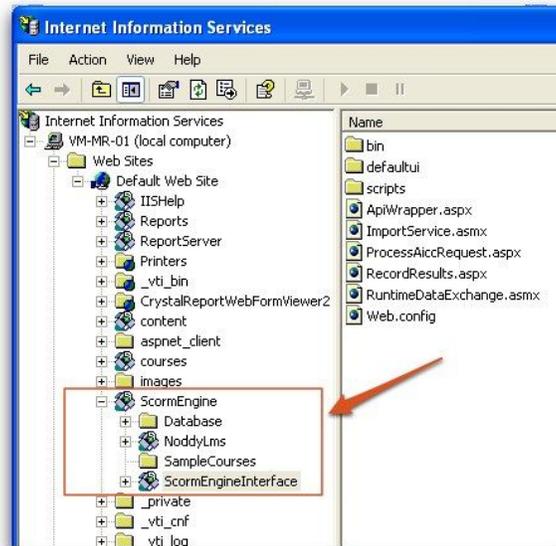


Click 'OK' to close the properties window and then re-open the properties window to by right clicking and selecting Properties (for some reason IIS requires you to close the window to commit the change which then enables the next step).

Click on the ASP.Net tab. From the ASP.Net version drop down, select a version of .Net that is 2.0 or greater. Then press 'OK' to exit.

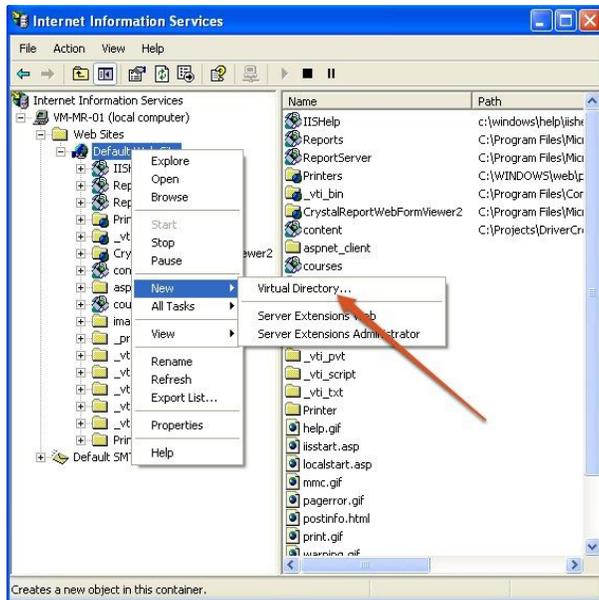


Repeat these two steps for the "NoddyLms" and "ScormEngineInterface" directories underneath the "ScormEngine" directory. When you are done, it should look like this:



If you put the files someplace other than the wwwroot directory, then you must create a virtual directory in IIS.

Right click on the "Default Web Site", mouse over "New", then select "Virtual Directory".



When prompted for an Alias, enter "ScormEngine".

When prompted for a Directory, browse to the directory where you copied the files and select the "ScormEngine" directory. If you copied the files to the "c:" drive, the directory path should be "c:\ScormEngine".

When prompted for permissions the default permissions do not need to be changed. The "Read" and "Run scripts" check boxes should be selected by default.

Now, follow the steps above (as if the files were placed in the wwwroot directory) to create applications and select ASP.Net v2.0+ for the three required directories ("ScormEngine", "NoddyLms" and "ScormEngineInterface"). Note, you will be able to skip the step of creating an application for the "ScormEngine" directory since virtual directories are automatically made into applications.

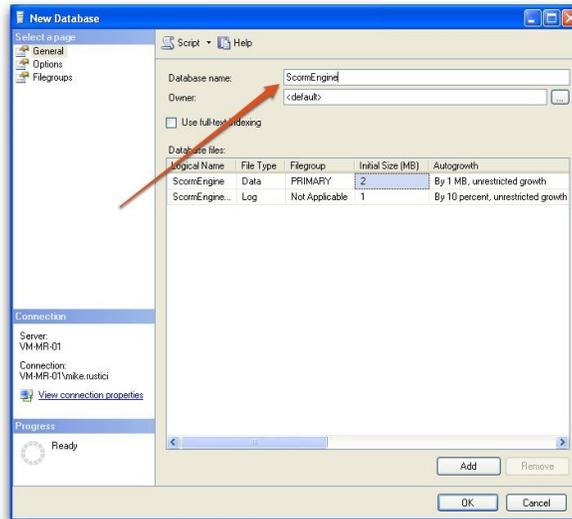
Step 3: Configure the database

The SCORM Engine can either be deployed into a new database or into an existing database. This document will describe how to create a new dedicated database, but these steps could just as easily be taken on an existing database.

Open SQL Server Management Studio. Right click on the Databases folder and select "New Database".



Enter a Database name of "ScormEngine" and leave the default values for the rest of the settings.



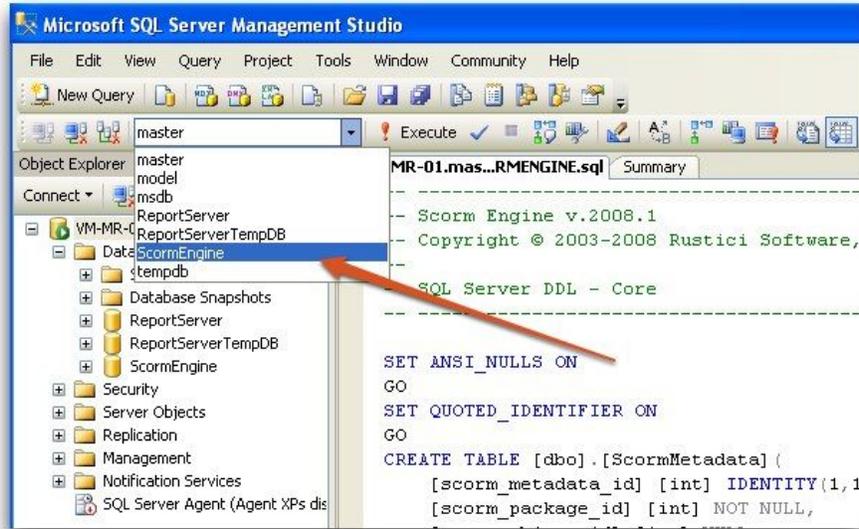
From the menu, select File->Open->File.



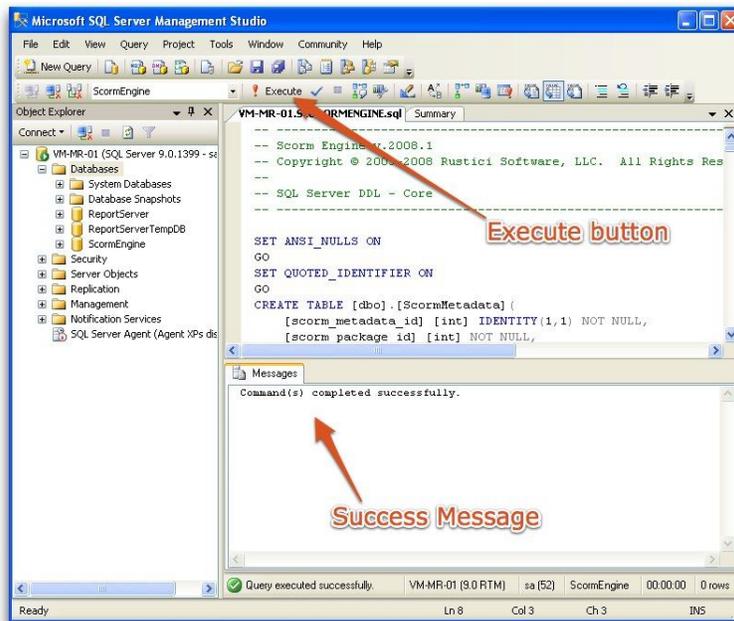
Browse to the directory where you copied the SCORM Engine files, open the "Database"

directory and then the "SqlServer" directory then select the file "1-2008.1_SQLSERVER_SCORMENGINE.sql". When prompted, connect using your standard login information.

Important: In the toolbar, ensure that the "ScormEngine" database is selected.



Run the script by pressing the "Execute" button in the toolbar. This will create all of the tables and other database objects needed to run the SCORM Engine. When the script is done, there should be a message in the Messages window stating "Command(s) completed successfully".



Repeat the steps to run the script named "2-2008.1_SQLSERVER_VANILLA INTEGRATION.sql". This script will create the files necessary for running the Noddy LMS.

There is also a third script in each database directory. This script can be run to remove all of the SCORM Engine and Noddy LMS database objects. That script can be useful for uninstalling the SCORM Engine when the database tables were installed into an existing database that can't simply be dropped.

Step 4: Configure the application

The SCORM Engine has a few settings that need to be configured for each installation. These settings are located in "ScormEngine/ScormEngineInterface/SCORMEngineSettings.config" file. Open this file in Notepad or your favorite text editor. Code editors such as Visual Studio can provide some helpful syntax highlighting that make editing the config file easier.

Database Connection String

The first setting that needs to be changed is the DatabaseConnectionString. Find the line called out in the image below.

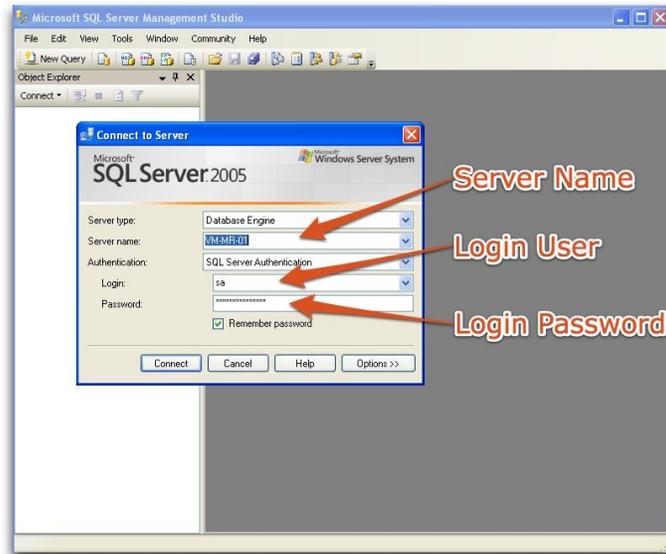


```
<?xml version="1.0" encoding="utf-8" ?>
<appSettings>
  <add key="LogicIntegrationAssemblyName" value="RusticiSoftware.ScormEngine.VanillaIntegration"/>
  <add key="LogicIntegrationClassName" value="RusticiSoftware.ScormEngine.VanillaIntegration.VanillaIntegr
  <add key="DataPersistenceEngine" value="sqlserver"/>
  <add key="DatabaseConnectionString" value="server=localhost;uid=sa;pwd=password;database=ScormEngine"/>
  <!-- Upload Import Control Default Integration Parameters -->
  <add key="WebPathToContentroot" value="/ScormEngine/NoddyLms/courses"/>
  <add key="FilePathToContentRoot" value="C:\Inetpub\wwwroot\ScormEngine\NoddyLms\courses"/>
  <add key="FilePathToUploadedZippedPackage" value="C:\Inetpub\wwwroot\ScormEngine\NoddyLms\uploads"/>
  <!-- URLs -->
  <add key="RedirectOnExitUrl" value="/ScormEngine/NoddyLms/NoddyLms.aspx" />
  <add key="UrlToCentralLaunchPage" value="/ScormEngine/ScormEngineInterface/defaultui/launch.aspx"/>
  <add key="UrlToUploadResources" value="/ScormEngine/NoddyLms/UploadImportControl/" />
```

The information in this string needs to be changed to match the login information for the database hosting the SCORM Engine. There are four essential pieces of information contained in this string:

1. The server name (defaulted to "localhost")
2. The database login user id (defaulted to "sa")
3. The database login password (defaulted to "password")
4. The database name (defaulted to "ScormEngine")

If you don't know this information, it can be derived from the connection screen that SQL Server Management Studio displays when it launches.



If the login information is disabled, you are likely connecting to the database using Windows Authentication instead of SQL Server Authentication. Try changing the value selected in the Authentication drop down. If you cannot connect to the database using SQL Server Authentication, you will need to login using Windows Authentication and [create a database login](#) for the ScormEngine.

The database name should be "ScormEngine" unless another name or database was used in Step 3.

Once you have determined the four values, replace the default values in the connection string setting with the values appropriate for your installation.

For example, if your values are:

1. Server name: "lmsdev1"
2. User id: "joeuser"
3. Password: "t0psecret"
4. Database Name: "ScormEngine"

The connection string setting would look like this:

```
<add key="DatabaseConnectionString" value="server=lmsdev1;uid=joeuser;pwd=t0psecret;database=ScormEngine"/>
```

File Paths

There are two settings that reference the installed location of the SCORM Engine. The "FilePathToContentRoot" and the "FilePathToUploadedZippedPackage" settings need to be updated to point to the place where the ScormEngine files were copied in Step 1.

```

<?xml version="1.0" encoding="utf-8" ?>
] <appSettings>
  <add key="LogicIntegrationAssemblyName" value="RusticiSoftware.ScormEngine.VanillaIntegration"/>
  <add key="LogicIntegrationClassName" value="RusticiSoftware.ScormEngine.VanillaIntegration.VanillaIntegr

  <add key="DataPersistenceEngine" value="sqlserver"/>
  <add key="DatabaseConnectionString" value="server=localhost;uid=sa;pwd=password;database=ScormEngine"/>

  <!-- Upload Import Control Default Integration Parameters -->
  <add key="WebPathToContentRoot" value="/ScormEngine/NoddyLms/courses"/>
  <add key="FilePathToContentRoot" value="C:\Inetpub\wwwroot\ScormEngine\NoddyLms\courses"/>
  <add key="FilePathToUploadedZippedPackage" value="C:\Inetpub\wwwroot\ScormEngine\NoddyLms\uploads"/>

  <!-- URLs -->
  <add key="RedirectOnExitUrl" value="/ScormEngine/NoddyLms/NoddyLms.aspx" />

  <add key="UrlToCentralLaunchPage" value="/ScormEngine/ScormEngineInterface/defaultui/launch.aspx"/>
  <add key="UrlToUploadResources" value="/ScormEngine/NoddyLms/UploadImportControl/" />

```

Installation path references

For example, if the ScormEngine was installed directly on the "c:" drive. The settings should look like this:

```
<add key="FilePathToContentRoot" value="C:\ScormEngine\NoddyLms\courses"/>
```

```
<add key="FilePathToUploadedZippedPackage"
value="C:\ScormEngine\NoddyLms\uploads"/>
```

Note that the "FilePathToContentRoot" setting should always point to the web directory referenced in the "WebPathToContentRoot" setting, which, by default, should point to the IIS directories created earlier.

Applying Changes

The SCORMEngineSettings.config file is a shared referenced configuration file that feeds the configuration settings normally stored in ASP.Net web.config file. Unfortunately, IIS has a bug that causes changes to the SCORMEngineSettings.config to not be recognized unless either IIS is restarted, or the web.config file in an application changes. To commit the changes made to the ScormEngineSettings.config file, either restart the IIS service or open the web.config files in the NoddyLms and ScormEngineInterface directories, make a small change and then delete the change (for instance, press space bar then backspace) then save the files.

Step 5: Permissions

For the SCORM Engine to operate properly, some system users need to have appropriate permissions over the ScormEngine directory.

Open My Computer and navigate to the ScormEngine directory. Right click on the ScormEngine directory and select Properties. Select the Security tab.

We need to ensure that three system users explicitly have permissions over this directory:

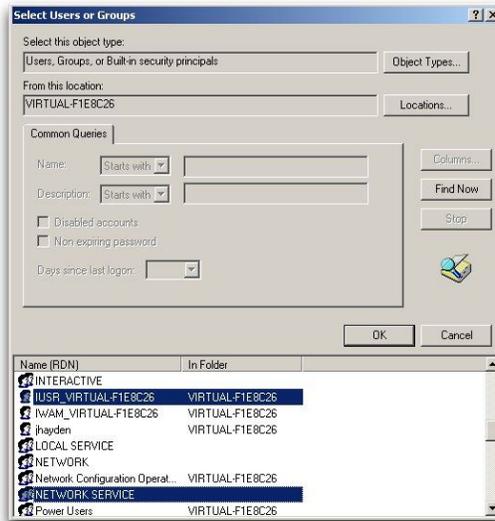
1. IUSR_machinename (where "machinename" is the name of your computer, for example "IUSR_LMSDEV1")
2. ASPNET
3. NETWORK SERVICE

If any of these users is not listed in the top text box "Group or user names:" (they probably will not be), they need to be added.

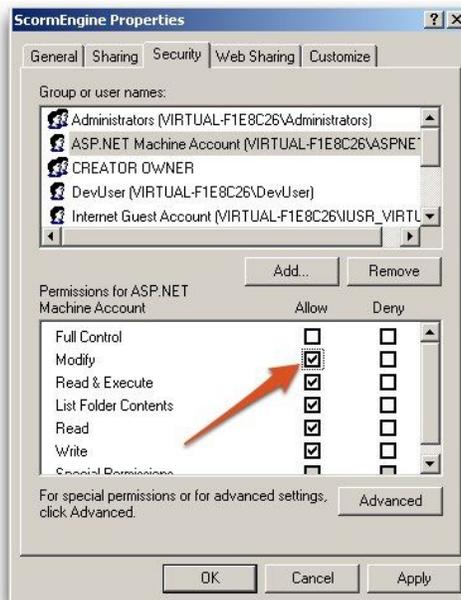


Click on the "Add" button. Ensure that the Location is the current computer name (if it is not, select the current computer after clicking on the "Locations" button). Click "Advanced" to bring up a dialog that lets you search for users.

If you have only a limited number of users on the computer, just click "Find Now" to list all of the users on the computer. Otherwise, enter some search criteria to limit the number of results. Select the three users listed above and press "OK".

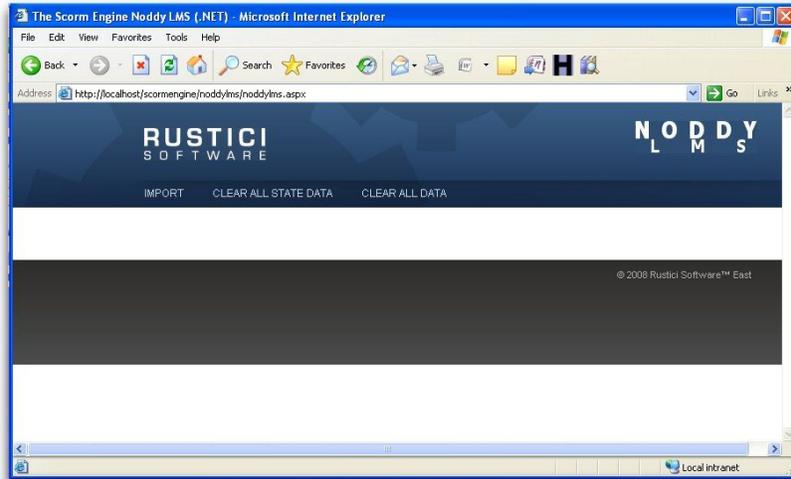


Click on each user and grant them Modify permissions by clicking on the check box in the "Allow" column next to "Modify". Granting "Modify" permissions, should also grant "Read & Execute", "List Folder Contents", "Read" and "Write".



Step 6: Launch

It's time to launch the Noddy LMS and make sure everything is functioning properly. Point your browser to http://SERVER_NAME/ScormEngine/NoddyLms/noddyLms.aspx. The screen below should appear. Click on Import and try importing one of the zip files in the SampleCourses directory. See the [Using the Noddy LMS](#) document for more information about what else you can do in the Noddy LMS.



Step 7: Clean up (optional)

In the SCORM Engine directory, the "Database" and "SampleContent" directories can be deleted once setup is complete. They are not necessary for running the SCORM Engine, but nor does it hurt to leave them there.

Instructions for Deploying the Noddy LMS for Java

This document will describe how to install the Java version of the SCORM Engine along with the Noddy LMS. It will describe a basic installation designed to get the SCORM Engine and Noddy LMS up and running quickly. These instructions were derived from an installation on Windows XP running the Apache HTTP Server connected to Apache Tomcat with mod_jk and using a MySQL database (with MySQL GUI tools installed). There are many permutations of platforms and application servers that can run the SCORM Engine and Noddy LMS. This document will take you through the basic steps that need to be performed on any platform, giving specifics for how to perform the installation on this particular platform

Prerequisites

This document assumes that the following are installed and properly configured on the target computer:

1. **Java Runtime Environment** (J2SE 5.0 or higher) - [JRE 1.6](#) used in this example
2. **Java Application Server** supporting J2EE 1.4 or higher (like Apache Tomcat, WebSphere®, JBoss® or WebLogic®) - [Apache Tomcat 6.0](#) using in this example.
3. **Web server** (like Apache HTTP Server or Microsoft IIS). Optional, if the application server is capable of serving HTTP - [Apache HTTP Server 2.2](#) used in this example.
4. **Database** (like MySQL, SQL Server or Oracle) - [MySQL 5.0](#) used in this example.
5. **Database JDBC Driver** (appropriate driver based upon Database selection) - [mysql-connector-java-5.1.6-bin.jar](#) used in this example.

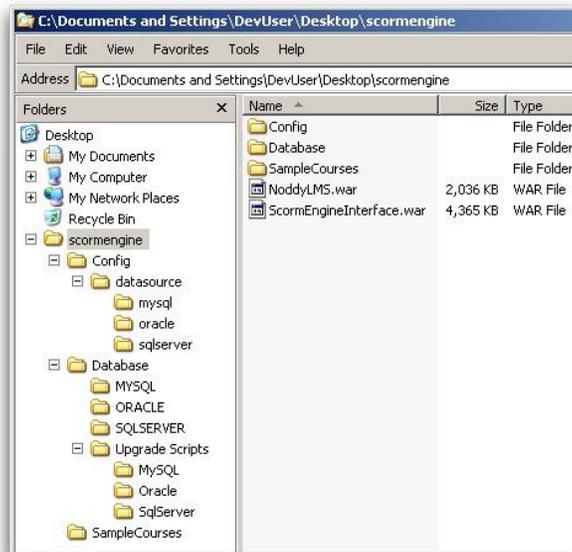
Summary

Here's the quick version of what needs to be done to install the SCORM Engine and Noddy LMS in our example configuration. {catalina.home} is the installation directory of Tomcat. (In a default installation on Windows, {catalina.home} is "C:\Program Files\Apache Software Foundation\Tomcat 6.0"). These steps are outlined in detail below.

1. Extract the files in your deployable to a temporary location.
2. Create a new database. Run the first two scripts (in order) listed in the Database folder of the deliverable to create the SCORM Engine's database objects.
3. Deploy the two *.war files from the root of the deployable to the Java Application Server. (By copying them to the "{catalina.home}/webapps/" directory in Tomcat).
4. If using a separate HTTP Server and Java Application Server, add entries to the application service connector for the two newly deployed *.war files. (By adding entries to the mod_jk file in the example configuration).
5. Create a JNDI Data Source to connect to the database created in Step #2. (By adding a resource entry to the context.xml file in the example configuration). Make sure your application server has a jdbc driver for your database. (By putting mysql-connector-java-5.1.6-bin.jar in "{catalina.home}/lib/" in the example configuration).
6. Copy the ScormEngineSettings.properties file to the web applications' class path. (The "{catalina.home}/lib/" is the simplest place to put it in the example configuration).
7. Modify the URL and file path settings in the ScormEngineSettings.properties file to point to your particular installation location. If the *.war files were deployed to the root of your application server, the only settings that should need to change are WebPathToContentRoot, FilePathToContentRoot and FilePathToUploadedZippedPackage. These should point to the "courses" and "uploads" directories within the NoddyLms application.
8. Restart the servers if needed to pick up the configuration changes and launch the application from http://SERVER_NAME/NoddyLms/NoddyLms.jsp.

Step 1: Extract the files

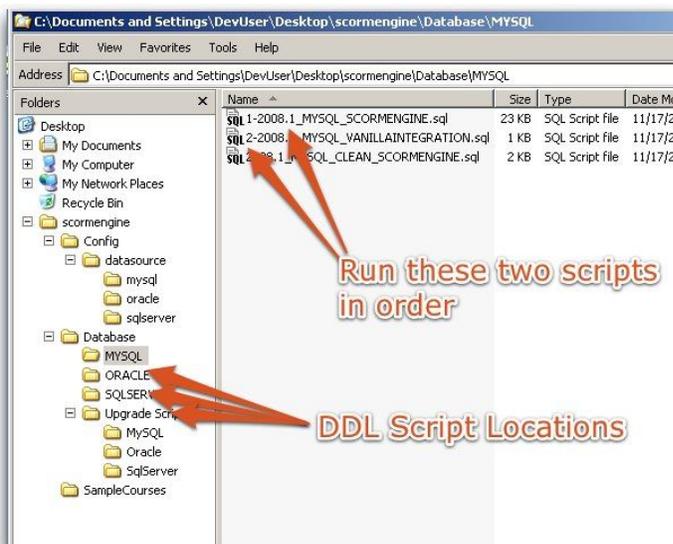
This should be a nice easy warm up step to get us started. Simply use your favorite compression utility to extract the zip file deployable to a temporary location. The resulting directory structure should look something like this:



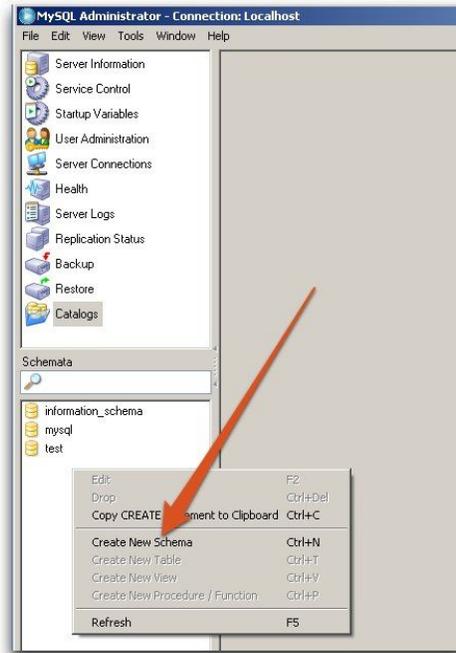
Step 2: Configure the database

The SCORM Engine can either be deployed into a new database or into an existing database. This document will describe how to create a new dedicated database, but these steps could just as easily be taken on an existing database.

The key aspect of this step is running the provided DDL scripts to create the tables, indices and other database objects required by the SCORM Engine. The DDL scripts are found in the "Database" folder within the deployable. Within that folder are subdirectories representing the scripts for the most commonly used database platforms (if your platform is not included in here, please let us know). The first two scripts should be run in order (they begin with "1" and "2" respectively). The third script will clean out the objects created by the first two scripts and should only be run if you wish to back out the changes made by the other scripts.

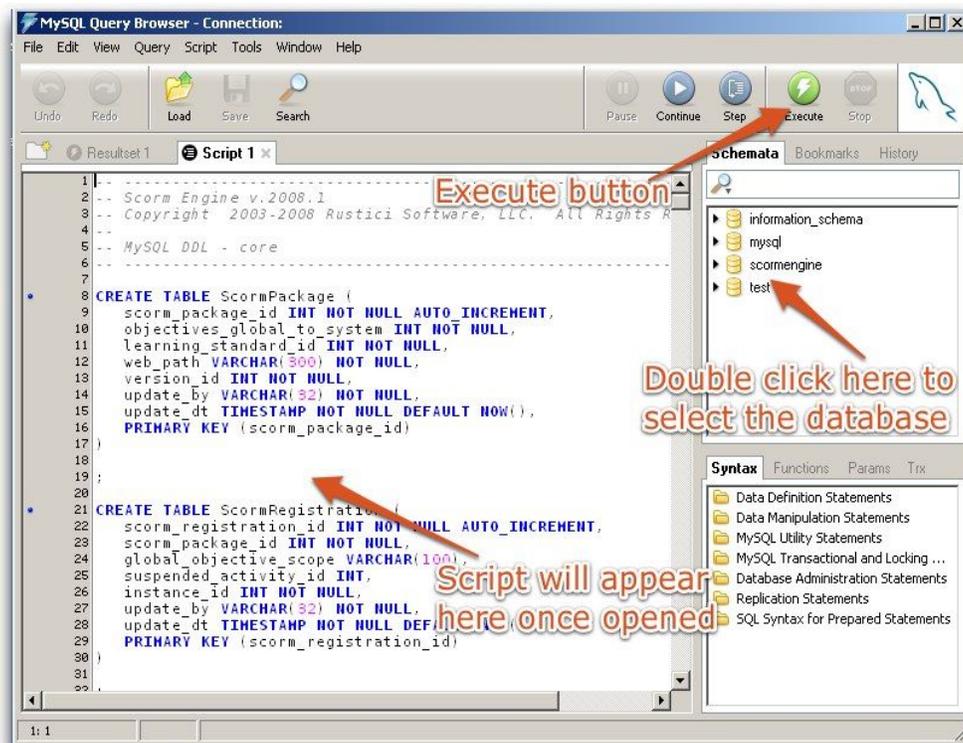


In MySQL, open up MySQL Administrator and select "Catalogs" from the list at the left. Right click in the list of databases at the bottom left and select "Create New Schema". We typically name the new schema "ScormEngine".



Now launch the MySQL Query Browser, either from the Start menu or from MySQL Administrator's Tools menu.

Select "File->Open Script..." and select the "{deployment_files_dir}/Database/MYSQL/1-2008.1_MYSQL_SCORMENGINE.sql" file. In the Schemata list on the right, double click on the ScormEngine database to select it and then press the Execute button to run the first script.



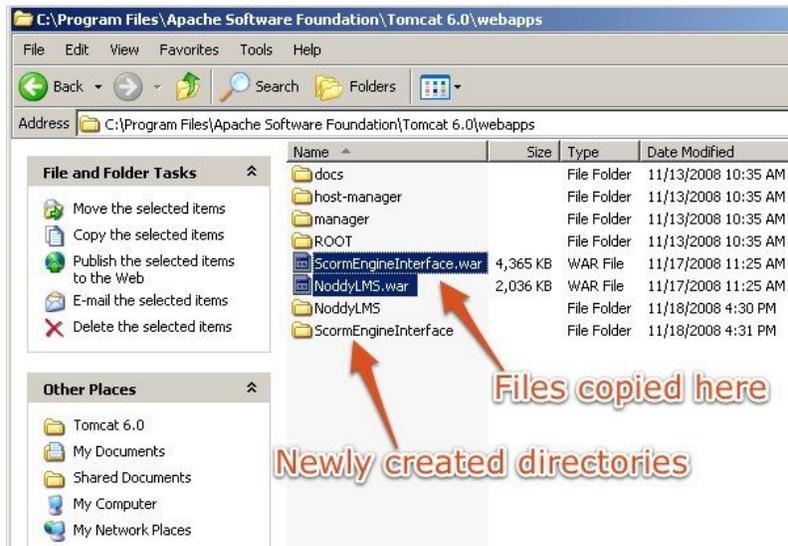
Repeat this step for the file
"{deployment_files_dir}/Database/MYSQL/2-2008.1_MYSQL_VANILLA INTEGRATION.sql".

Step 3: Deploy the *.war files

This step will vary considerably based on the Java Application Server being used. You will need to use the two provided *.war files to create new applications on your application server. You may name them whatever you wish, but for simplicity it is best to deploy them to the root of the server and leave their names as "NoddyLms" and "ScormEngineInterface". If alternate locations are chosen, there will be additional configuration required in Step #7.

In Apache Tomcat, creating a new application from *.war files is easy. By default, all you have to do is copy the *.war files into the "{catalina.home}/webapps" directory. If the Tomcat server is running, it will automatically detect the new *.war files and create the applications. Simply copy "NoddyLms.war" and "ScormEngineInterface.war" from the root of the deployment files directory to the "{catalina.home}/webapps directory".

After a moment, Tomcat will create two new application directories in the webapps folder. This will indicate that Tomcat has successfully installed the new web applications.



Step 4: Connect the web server with the new Java applications

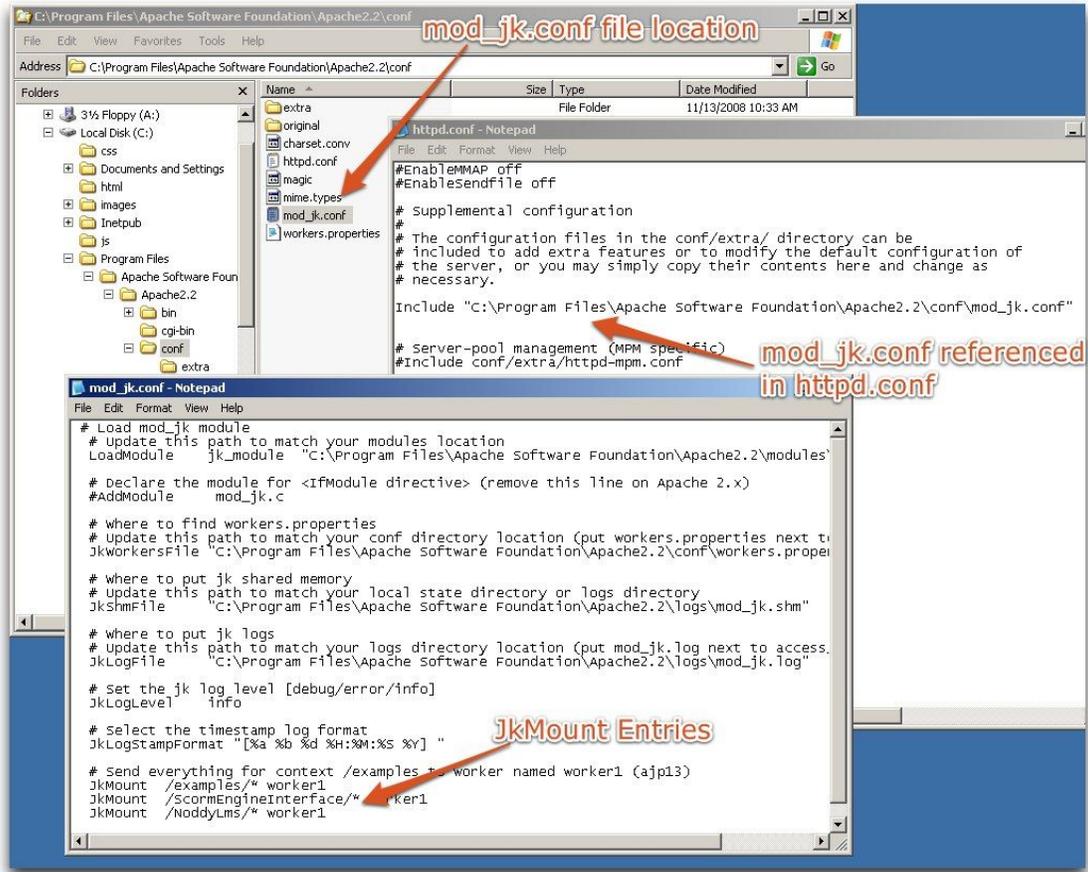
If using a web server that is separate from the Java Application Server you will need to inform the web server of the newly created Java applications. The web server needs to know where on the application server to redirect requests for URLs in the NoddyLms and ScormEngineInterface applications.

In the example configuration, this link is established by adding "JkMount" entries to the Apache configuration. These entries are often included in a mod_jk.conf file (that is referenced from the httpd.conf file) or they are included directly in the httpd.conf file. This example has a mod_jk.conf file that is referenced from the httpd.conf file, both of which are located in the "conf" dir within the Apache home directory.

The entries that should be added to the configuration are:

```
JkMount /ScormEngineInterface/* worker1
```

JkMount /NoddyLms/* worker1"worker1" might be replaced with the name of worker in your instance of Tomcat. Workers are defined in the file specified by the "JkWorkersFile" entry in the mod_jk config (likely right above the entries for "JkMount").



Step 5: Create a JNDI Data Source

The Java Application Server needs a JNDI Data Source to connect to the database created in Step #2. The data source should be named "jdbc/ScormEngineDB".

In Apache Tomcat, a JNDI Data Source is created by creating a new "Resource" entry in the context.xml file located in the "{catalina.home}/conf" directory. Example resource entries are located in the "Config/datasource" folder within the deployable. Find the appropriate example for your database, open the XML file in a text editor and copy the entire "Resource" tag to the clipboard (don't forget to include the "</>" characters at the end). Now, open the context.xml file in the "{catalina.home}/conf" directory in a text editor and paste the example "Resource" tag anywhere within the "Context" tag. Update the values in the username, password and url attributes to point to the database created in Step #2 with the appropriate login credentials.

```
See the License for the specific language governing permissions and
limitations under the License.
-->
<!-- The contents of this file will be loaded for each web application -->
<Context>

  <!-- Default set of monitored resources -->
  <WatchedResource>WEB-INF/web.xml</WatchedResource>

  <!-- Uncomment this to disable session persistence across Tomcat restarts -->
  <!--
  <Manager pathname="" />
  -->

  <!-- Uncomment this to enable Comet connection tacking (provides events
  on session expiration as well as webapp lifecycle) -->
  <!--
  <Valve className="org.apache.catalina.valves.CometConnectionManagerValve" />
  -->

  <Resource name="jdbc/ScormEngineDB" auth="Container"
    type="javax.sql.DataSource"
    description="MySQL database for SCORM engine"
    username="root"
    password="password"
    driverClassName="com.mysql.jdbc.Driver"
    url="jdbc:mysql://localhost/ScormEngine?autoReconnect=true"
    maxActive="30"
    maxIdle="0"
    maxWait="10000"
    validationQuery="Select 1"
    testOnBorrow="true"
    testWhileIdle="true"
    timeBetweenEvictionRunsMillis="10000"
    maxEvictableIdleTimeMillis="60000"
  />
</Context>
```

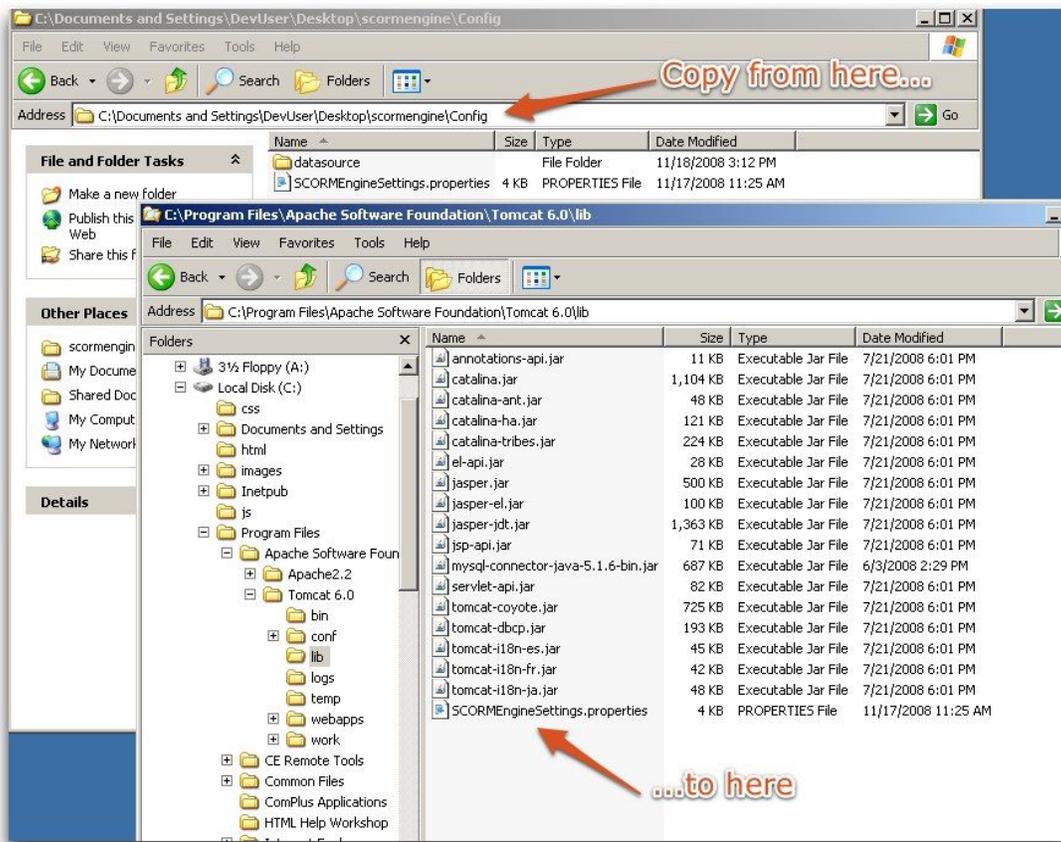
The diagram features two callout boxes with orange text and arrows. The first box, labeled "Values to update", has arrows pointing to the configuration parameters: `username="root"`, `password="password"`, and `url="jdbc:mysql://localhost/ScormEngine?autoReconnect=true"`. The second box, labeled "'Resource' tag placed within the 'Context' tag", has arrows pointing to the opening and closing tags of the `<Resource>` element.

Make sure your application server has a jdbc driver for your database. (By putting `mysql-connector-java-5.1.6-bin.jar` in `"{catalina.home}/lib/"` in the example configuration).

Step 6: Deploy the ScormEngineSettings.properties file

The `SCORMEngineSettings.properties` file contains a number of configuration and deployment settings that the SCORM Engine relies on. The `SCORMEngineSettings.properties` file is located in the "Config" directory of the deployable. This file needs to be copied to somewhere in the web applications' class path.

In the example configuration, the `"{catalina.home}/lib/"` directory is the simplest place to put this properties file.



Step 7: Edit the settings in the SCORMEngineSettings.properties file

Open the newly deployed SCORMEngineSettings.properties file in a text editor. Some of the settings in this file will need to change to be consistent with your deployment.

In the "URLS" section of the configuration, nothing will need to change IF you deployed the *.war files to the root of your application server in Step #3. If you deployed the files to an alternate location, all of the URLs in this section will need to be updated to refer to the location of your deployment.

The settings in the section titled "Upload Import Control Default Integration Parameters", control where course files are placed when they are uploaded during the import process. There are two locations that need to be recorded, "FilePathToUploadedZippedPackage" is a path on the file system where zip files containing courses are uploaded temporarily before they are extracted. Once courses are extracted, they are placed in "FilePathToContentRoot". This directory is where courses are ultimately server from by the web server.

"WebPathToContentRoot" specifies the URL that the web server maps to the "FilePathToContentRoot" directory. In a production system, you will likely want to customize these settings to properly integrate with your existing content deployment scheme (your integration consultant can help you with this process). To simply deploy the Noddy LMS and get things up and running, you can set these values to point to the "courses" and "uploads" directories included in the NoddyLms.

WebPathToContentRoot: "/NoddyLms/courses"

FilePathToContentRoot: "{catalina.home}/webapps/NoddyLms/courses"

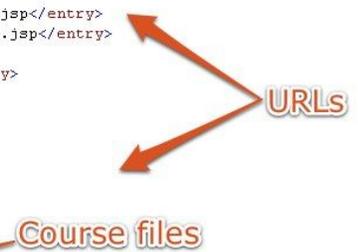
FilePathToUploadedZippedPackage: "{catalina.home}/webapps/NoddyLms/uploads"

```
<entry key="DataHelperAssemblyName">RusticiSoftware.ScormContentPlayer.DataHelp</entry>
<entry key="DataHelperClassName">RusticiSoftware.ScormContentPlayer.DataHelp.JdbcDataHelper</entry>
<entry key="DatabaseConnectionString">n/a</entry>

<!-- URLs -->
<entry key="RemoteAiccRequestProcessorUrl">/ScormEngineInterface/ProcessAiccRequest.jsp</entry>
<entry key="CentralAiccRequestProcessorUrl">/ScormEngineInterface/ProcessAiccRequest.jsp</entry>
<entry key="ScormResultProcessorUrl">/ScormEngineInterface/RecordResults.jsp</entry>
<entry key="UrlToCentralLaunchPage">/ScormEngineInterface/defaultui/launch.jsp</entry>
<entry key="RemoteLaunchPageUrl">/ScormEngineInterface/defaultui/</entry>
<entry key="ScormEngineScriptsUrl">/ScormEngineInterface/scripts</entry>
<entry key="StylesheetUrl">/ScormEngineInterface/defaultui/defaultstyles.css</entry>
<entry key="RedirectOnExitUrl">/NoddyLms/NoddyLms.jsp</entry>
<entry key="CentralWebServiceUrl"></entry>
<entry key="ImportWebServiceUrl"></entry>

<!-- Upload Import Control Default Integration Parameters -->
<entry key="WebPathToContentRoot">/NoddyLms/courses</entry>
<entry key="FilePathToContentRoot">C:\Program Files\Apache Software Foundation\Tomcat 6.0\webapps\NoddyLms\courses</entry>
<entry key="FilePathToUploadedZippedPackage">C:\Program Files\Apache Software Foundation\Tomcat 6.0\webapps\NoddyLms\uploads</en

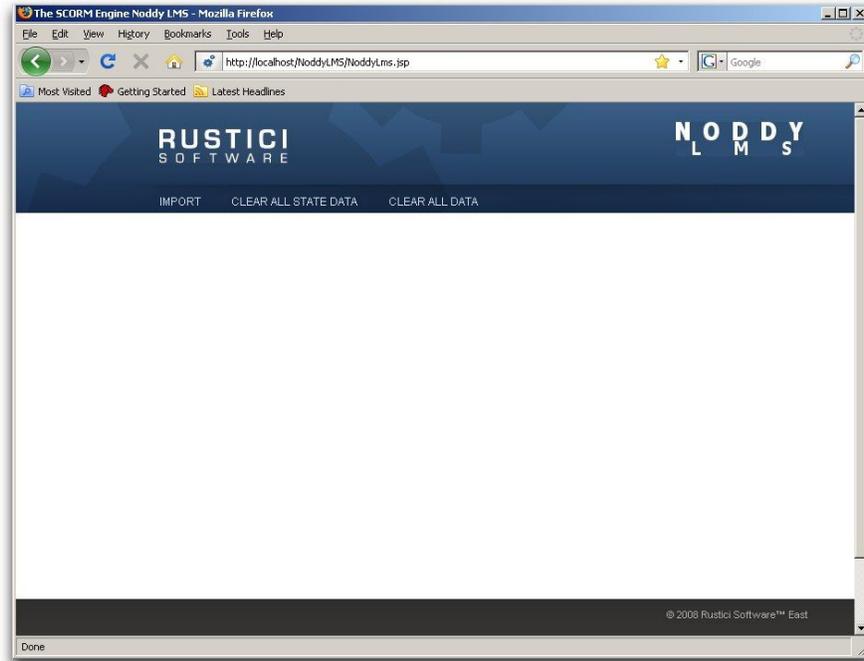
<!-- REGISTRATION RESTART
1 = "never":
```



The diagram shows two red arrows pointing from the code to labels. One arrow points from the 'URLs' section of the code to the label 'URLs'. The other arrow points from the 'WebPathToContentRoot' and 'FilePathToContentRoot' entries to the label 'Course files'.

Step 8: Restart and launch

Restart the web server and Java Application Servers if needed to pick up the configuration changes and launch the application from http://SERVER_NAME/NoddyLms/NoddyLms.jsp. The screen below should appear. Click on Import and try importing one of the zip files in the SampleCourses directory. See the [Using the Noddy LMS](#) document for more information about what else you can do in the Noddy LMS.



SCORM Engine Settings

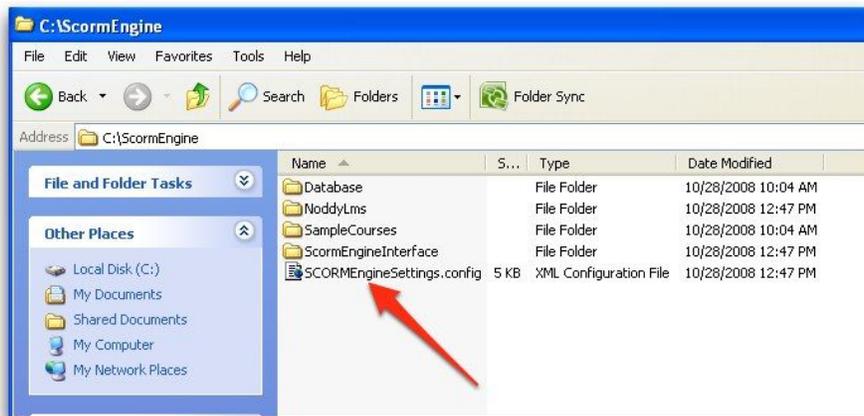
The SCORM Engine contains a number of configuration settings. These settings contain logistical information about how the SCORM Engine is deployed and they control how the SCORM Engine behaves. The SCORM Engine settings do not need to be changed frequently. They are typically only accessed during integration 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

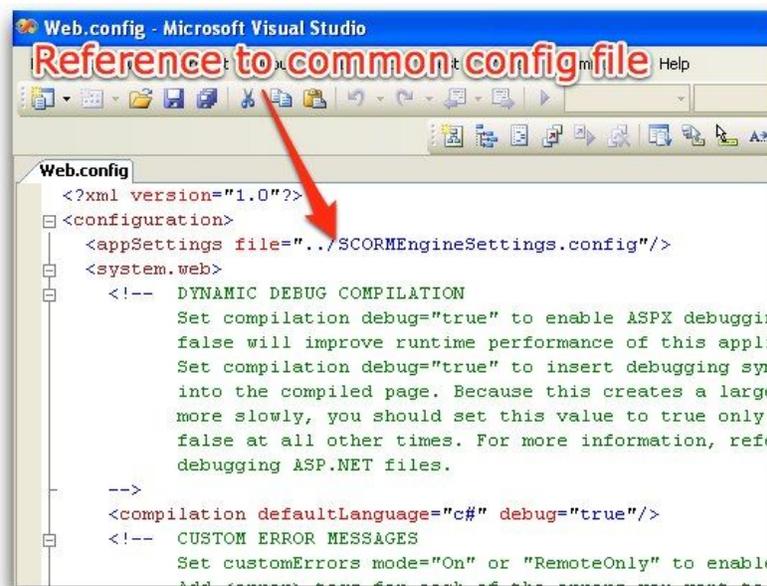
The method for accessing and changing the SCORM Engine settings varies depending on the platform you are running (.NET or Java).

.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.

```

SCORMEngineSettings.config
<?xml version="1.0" encoding="utf-8" ?>
<appSettings>
  <add key="LogicIntegrationAssemblyName" value="RusticiSoftware.VanillaIntegration"/>
  <add key="LogicIntegrationClassName" value="RusticiSoftware.ScormContentPlayer.VanillaIntegr

  <add key="DataPersistenceEngine" value="sqlserver"/>
  <add key="DatabaseConnectionString" value="server=localhost;uid=sa;pwd=password;database=Sc

  <!-- Upload Import Control Default Integration Parameters -->
  <add key="WebPathToContentroot" value="/ScormEngine/NoddyLms/courses"/>
  <add key="FilePathToContentRoot" value="C:\ScormEngine\NoddyLms\courses"/>
  <add key="FilePathToUploadedZippedPackage" value="C:\ScormEngine\NoddyLms\uploads"/>

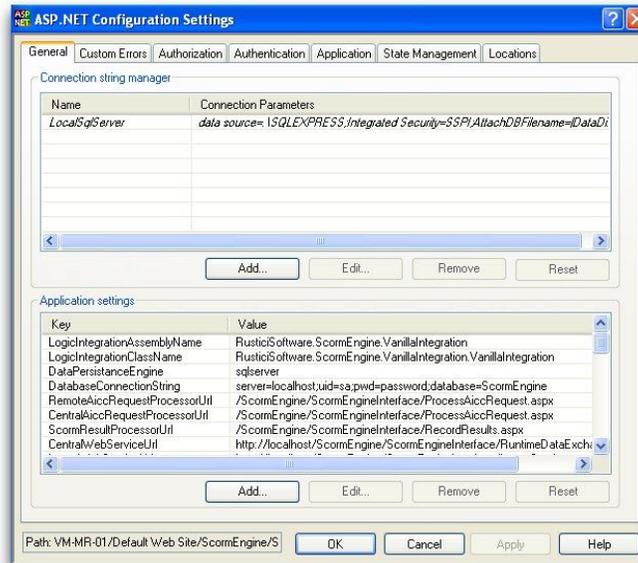
  <!-- URLs -->
  <add key="RedirectOnExitUrl" value="/ScormEngine/NoddyLms/NoddyLms.aspx" />

  <add key="UrlToCentralLaunchPage" value="/ScormEngine/ScormEngineInterface/defaultui/launch.
  <add key="UrlToUploadResources" value="/ScormEngine/NoddyLms/UploadImportControl"/>

  <add key="StylesheetUrl" value="defaultstyles.css"/>

```

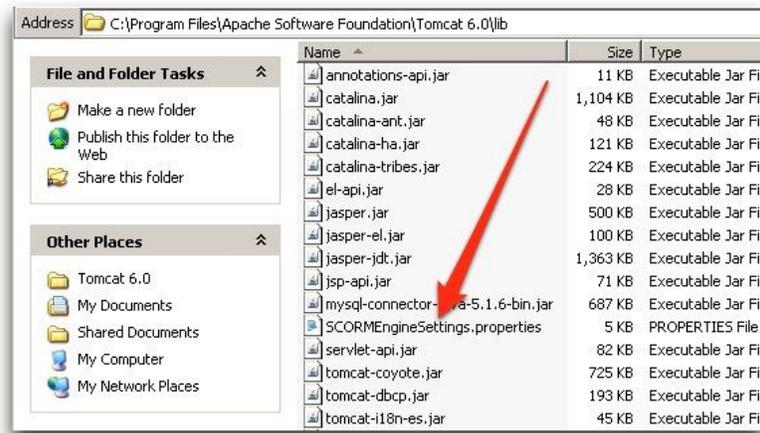
Alternatively, recent version of IIS include an "Edit Cofiguration" button on the ASP.NET tab of the applications properties. This button brings up a GUI for editing application settings individually.



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:

1. Integration class
2. Data persistence
3. URLs
4. Upload import control
5. Registration instance and package versioning
6. Optional SCORM Engine features
7. Debug settings
8. 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.
"ole"	Connect to any database that supports an OLE interface . (.NET Only)
"plugin"	Use a custom developed 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.

Note: The "not _optimized" versions of the Oracle and SQL Server persistence engines are recommended for high volume installations. The standard versions are more efficient for lower volume installations.

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;UserId=ScormEngine;Password=password;Integrated Security=no"
IBM DB2	"Provider=IBMDADB2;DataSource=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 persistence classes and should be used instead of these settings unless an uncommon scenario is encountered.

DataPersistenceAssemblyName (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"

DataPersistencePersistClassName (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 manipulating SCORM Engine data within the database.

Example: "RusticiSoftware.ScormContentPlayer.Logic.SqlPersistImplementation"

DataPersistenceRetrieveClassName (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 manipulating 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"

DataPersistenceUseStoredProcsIfAvailable (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".

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.

RemoteAiccRequestProcessorUrl - 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.aspx" file. This setting is only needed for cross domain, central/remote deployments. This URL should be an fully qualified URL to the "RuntimeDataExchange.aspx" 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.aspx"

ImportWebServiceUrl - URL to the "ImportService.aspx" 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.aspx" 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.aspx"

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. Note: This URL should include a slash at the end.

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. Note: This URL should not include a slash at the end.

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"`

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".

Note: This setting should include a slash at the end.

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. Note: This setting should not include a slash at the end.

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. Note: This setting should not include a slash at the end.

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. Note: This setting should include a slash at the end.

Example: `"/ScormEngine/NoddyLms/UploadImportControl/"`

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).

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)

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 happened 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 were 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).

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).

Central / Remote Architecture

These settings apply to the use of the cross domain, central/remote architecture.

UseWebServices - 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 communication with the central instance has been lost and notify the user that an error has occurred. 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 $\text{WebServicesRetries} * \text{WebServiceRetryInterval}$) should be significantly less than the default `CommCommitFrequency` 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)

SCORM Engine Launch Parameters

When launching the SCORM Engine, there are 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 valid 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 registration 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 parameter 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 parameter 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 registration with the data model element for credit ("cmi.credit" or "cmi.core.credit") set to "no credit". This setting is useful for launching courses that should be tracked but that don't "count" for anything. If not included, the default value for this parameter 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 objects is 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 value 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 representation of "|" is "%7c". Once escaped, the above example would look like:

userName%7cjoeuser!courseId%7c42The Noddy LMS can simplify the process of creating 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.

CourseId **1001** **Photoshop Example -- Linear** New Registration Update Package Properties Delete 1 version 1 registration

Create a new registration by filling in the values and clicking launch below. X

UserName

CourseId

Launch URL
/ScormEngine/ScormEngineInterface/defaultui/launch.aspx?configuration=®istration=UserName|joeUser|CourseId|1001

Serialized value 

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

"regForCreate" 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.

"registrati on" A serialized external registration id
"package" Not included
"manifest DirPath" Not included
"webPath" Not included
"
"tracking" "false"
"forceRevision" "true"
"regForCredit" Not included
edit"

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.
"registrati on" Not included
"package" A serialized external package id
"manifest DirPath" Not included
"webPath" Not included
"
"tracking" Not included (only relevant if a registration is passed in)
"forceRevision" Not included
"regForCredit" Not included
edit"

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.
"registrati on" A serialized external registration id

"package" A serialized external package id
"manifest DirPath" Not included
"webPath" Not included
"
"tracking" Not included
"forceReview" Not included
"regForCredit" "false"
edit"

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
"manifest DirPath" File path to manifest
"webPath" Web path to course directory
"
"tracking" Not included
"forceReview" Not included
"regForCredit" Not included
edit"

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 2008.1 Upgrade Notes

Significant Integration Changes

The "central" and "remote" web applications have been consolidated.

The SCORM Engine supports an advanced deployment scenario we term the "remote/central architecture". The majority of our clients do not take advantage of the advanced content deployment strategies this architecture enables so we decided to consolidate our code base to simplify deployment for the majority. Previously the SCORM Engine deployable included two separate applications, a "central" and a "remote" that had to be deployed in all instances. In this release, these projects have been consolidated into a single application called "SCORM Engine" This web application contains all functionality and in most instances will be the only web application that is deployed. For installations which use the "remote/central architecture", the single SCORM Engine application will be deployed in their place. In this case, configuration settings will determine if the installation should take on a "central" role or a "remote" role. The configuration settings will be the same as they were when separate applications were used.

The SCORM Engine libraries have been consolidated into a single library.

Rather than deploying seven separate code libraries as we did previously, the SCORM Engine now includes a single library file named RusticiSoftware.ScormEngine.dll (.Net) or scormengine.jar (Java). This change will allow Rustici Software to better control versioning and upgrades of the core SCORM Engine and simplify the integration development environment. For clients who are linking directly to the SCORM Engine's code, you are encouraged to reference this code library directly rather than incorporating the source code into your solutions and compiling it on your own. We will still provide full source code access, but this change will allow us to provide you with better support since we will have more precise knowledge about what version of our code you are running against.

A new default UI has been created.

For installations that have done little customization to the "skins" (*deliver.aspx, *launch.aspx, etc.), it is recommended that you switch to using the default UI. The stylesheet for this default UI provides a degree of customization and will allow Rustici Software to provide better management of the skin going forward.

A formalized public API was added.

A ScormEngineManager class has been added to formalize the public API into the SCORM Engine that LMS integrations should be using.

There is a new ScormEngineManager class that provides a clean formal API for interacting with the SCORM Engine's business logic. This class provides a unified way to add/get/delete/update course packages and registrations, using only the external keys that the external LMS is knowledgeable of. Furthermore, all methods in the ScormEngineManager are version/instance aware so clients can more appropriately make use of the new built-in package versioning and multiple registration instance capability (if enabled).

Upgrading from SCORM Engine 2007.1

Step 1: Change your library references

With SCORM Engine 2008.1 onward we will be delivering a versioned binary dll/jar which should be used in place of RusticiSoftware.ScormContentPlayer.* source code projects in your solution. Please remove these projects and add remove your library references to RusticiSoftware.ScormContentPlayer.* projects. In their place you will simply need to add a reference to RusticiSoftware.ScormEngine.dll (.NET) or scormengine.jar (Java). You will still need your integration source code project as well as the web application that provides import and related functionality.

Step 2: Utilize the new SCORM Engine web application

If your ScormContentPlayerCentral and ScormContentPlayerRemote web applications were previously deployed directly to IIS as independent web applications, you will simply need to remove those and add the new ScormEngine web application. If the SCORM Engine files are deployed directly beneath the IIS root (typically c:\inetpub\wwwroot) you will just need to browse to the directory in IIS and click the "create" application button under its properties. Otherwise, you will need to create a new virtual directory named ScormEngine in IIS and point it to the SCORM Engine application files.

Finally, copy your freshly compiled integration library into the ScormEngine's /bin directory.

Some LMS integrations actually choose to include the ScormEngine as a subdirectory of their own LMS web application. If that's the case, you will need to remove your references to the RusticiSoftware.ScormContentPlayer.* projects and remove the "central" and "remote" application files from your web application. In their place, copy the entire ScormEngine web application to a subdirectory named \ScormEngine. You can do this by simply dragging the \ScormEngine folder directly from Windows Explorer into the Visual Studio Solution Explorer. The aspx files will be treated as content rather than source code. Then add a reference to the RusticiSoftware.ScormEngine.dll that's in \ScormEngine\bin. Finally, add an entry to your web.config to include \ScormEngine\bin in the dll search path

```
<configuration>  
  <runtime>
```

```

        <assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
            <probing privatePath="ScormEngine\bin"/>
        </assemblyBinding>
    </runtime>
</configuration>

```

Step 3: Upgrade your Database

Run the appropriate 2007.1 -> 2008.1 upgrade script for your particular database (SQL Server, Oracle, MySQL, DB2).

Step 4: Configuration Changes

A few new configuration settings have been added. Listed below are the new settings with the default settings. Note that the versioning parameters are only useful to those integrations which are created to take advantage of them -- simply changing the parameter will not automatically enable versioning. Similarly, the SSP settings are only applicable to those clients who have licensed the SSP plugin. These settings are typically found in the ScormEngineSettings.config file or directly in our application's web.config file.

```

    <add key="DataPersistenceUseStoredProcsIfAvailable" value="true"/>

    <add key="SSPEnabled" value="true" />
    <add key="SSPSizeAllocation" value="1048576"/>

    <add key="StylesheetUrl" value="/scormengine/defaultui/scormengine.css"/>

    <add key="WhenToRestartRegistration" value="never" />
<add key="IsPackageVersioningEnabled" value="false"/>

```

Step 5: Integration Class Changes

If you have made no customizations to your deliver UI we would suggest you utilize the new default UI now included as part of the core SCORM Engine. To do this, simply remove the GetCosmeticInfo() methods from your integration class so the default implementations will be used. Then change your launch string to point to /ScormEngine/defaultui/launch.aspx. Some degree of customization is still available through the use of a CSS Stylesheet. The url for this stylesheet is in your ScormEngineSettings.config file (or web.config).

Step 6: Web Controls References Change

Because all library code, including web controls, has been consolidated into RusticiSoftware.ScormEngine.dll you will need to make a small change to any .aspx pages to utilize the SCORM Engine web controls. Simply change all references in .aspx pages from Assembly="RusticiSoftware.ScormContentPlayer.WebControls" to Assembly="RusticiSoftware.ScormEngine".

SCORM Engine Course Properties

[Documentation no longer available for this section]

SCORM Engine Scalability

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:

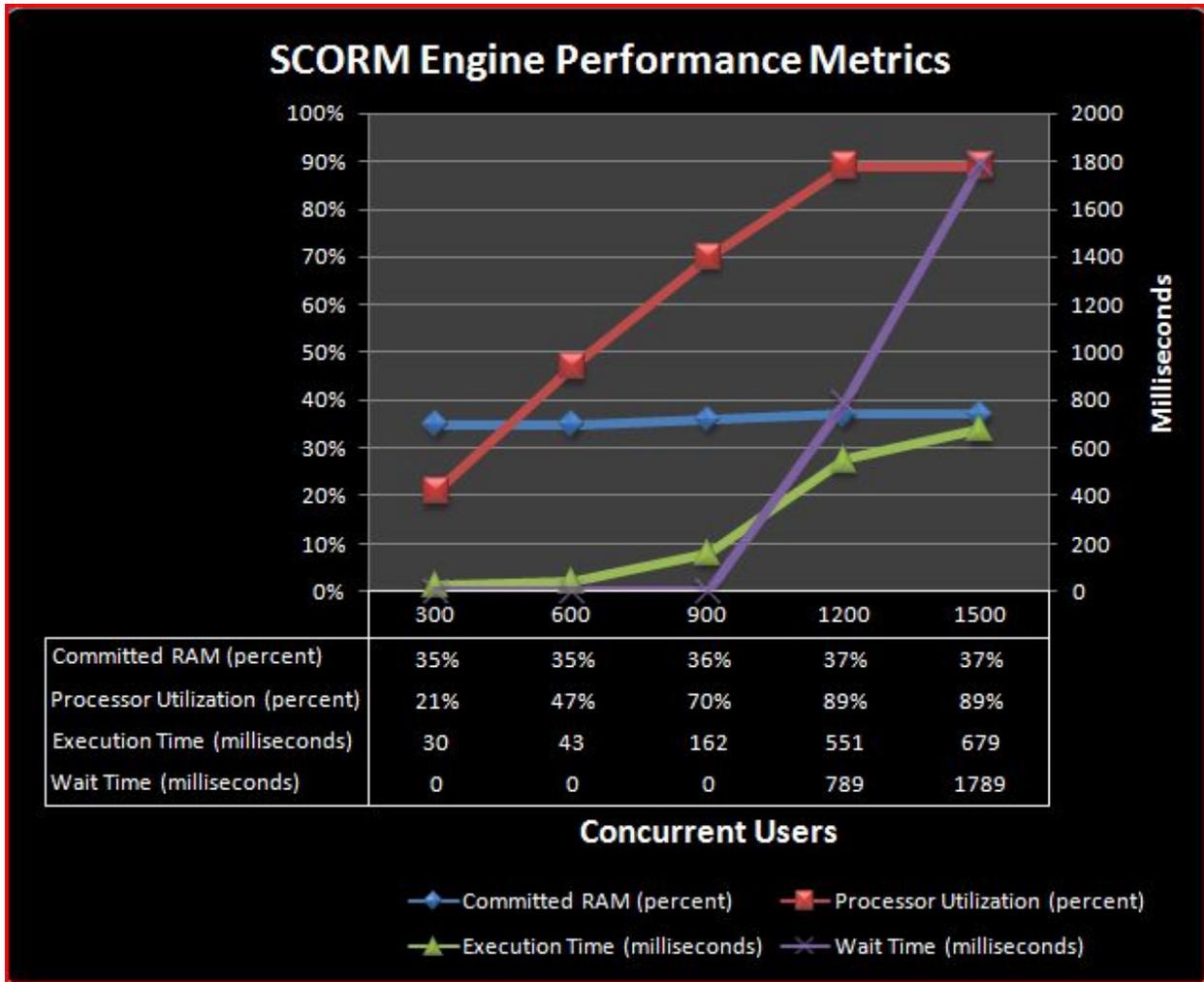
- Processor Utilization – Percentage of available processor time used by the application
- Committed RAM – Percentage of available RAM used by the application
- Wait Time – The amount of time the HTTP requests waiting in a queue before they were processed.
- Execution Time – The amount of time it took 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.

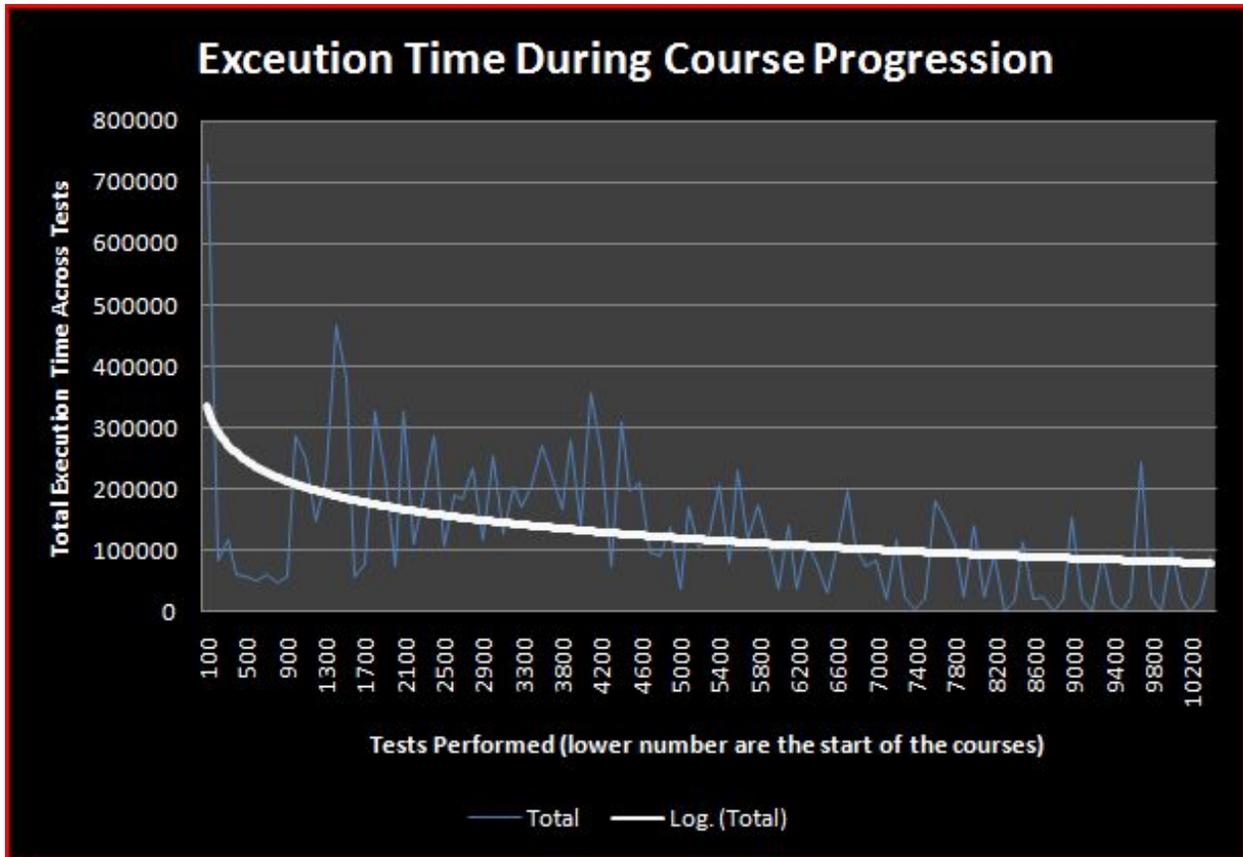
Results

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.