Sahara Development Handbook

written for

LabShare

at

University of Technology Sydney
Faculty of Engineering and Informaion Technology

Author: Tania Machet
Co-authors: Michael Diponio
Tejaswini Deshpande

Sydney, September 2010

Contents

Document Control Sheet

Contact for enquiries and proposed changes

If you have any questions regarding this document, or if you have any suggestions for improvements, please contact:

Document owner: Tania Machet
Email: tania.machet@eng.uts.edu.au

Version No.	Date	Changed by	Nature of Ammendment
0.1	23/08/2009	Tania Machet	Initial draft
0.2	27/01/2010	Tania Machet	Update for Ch 4. and 5
0.3	02/02/2010	Tania Machet	Update after review and corrections
0.4	04/03/2010	Tania Machet	Update from code changes and additional information
1.0	08/03/2010	Tania Machet	Installation guide update and baseline
1.1	12/03/2010	Tania Machet	Update for service wrapper
1.2	07/06/2010	Tania Machet	Change title, include Web Development and Primitive control
1.3	23/08/2010	Tania Machet	Updated after reviews

Chapter 1

Introduction

1.1 Purpose of this document

This document is intended for use by developer wishing to integrate computer-controlled and remotely-accessed rigs into Sahara. It is a technical document that describes what development is needed for a Sahara *Rig Client* so that a rig can be accessed using the Sahara *Web Interface and managed with the Sahara* Scheduling Server. It also covers the development required to customise the Sara Web Interface to site and rig specific requirements.

The development described here is in PHP, JavaScript and Java and the document assumes some technical knowledge of these.

The document details the steps a developer should follow to determine what Rig Client development is needed for their specific rig. It then provides instructions and examples of what needs to be developed for each of the Rig Client types as well as how to develop a customised interface for your rig that users will see when accessing it through Sahara.

Also incuded is a description of how the user can customise the Web Interface for their site. This includes authentication, branding (eg page headers) and deployment specific information (eg contact information)

1.2 Definitions

Term	Definition
Action	Rig specific implementation of a single behaviour
Classpath	Java Virtual Machine, a set of software that uses the virtual machine model to execute other programs and scripts. JVMs accept Java byte code (usually generated from Java Source code). Allows for the "write once, run anywhere" aspect of Java.
JNI	Java Native Interface, the is a programming framework that allows Java code running in a JVM to call and to be called by programs specific to a hardware and operating system platform (native programs) and libraries written in other languages.
Master User	The user who initiates the session and can terminate the session. Also have complete control of the rig.
Slave Active User	The user who is assigned access to an existing rig session. An active slave user has similar access to the rig as the master session user, but may not terminate a session and may not assign slave users.
Slave Passive User	The user who is assigned access to an existing rig session. A passive slave has limited access to the experiment and implementations should interpret this permission as a read-only user (can view the experiment control interface and audio/visual output, but may not control the rig).
Interactive Control	This is where the rig is controlled by a user who is physically present during the session and executing control on the rig.
Batch Control	Control is done by means of an executable file that contains instructions on controlling the rig. rUsing this type of control, the user does not have to be present while the session exists.

Chapter 2

Sahara Overview

2.1 History

A discussion of the prevous remote sharing software, why we changed and why we chose to do it this way

2.2 What is Sahara?

A "'theory of operation"' - what Sahara is, how to develop etc

2.3 Components

Sahara is a collection of software packages that allows a remotely control lab to be built using computer controlled equipment. This allows a large number of remote users to equitably access a limited number of rigs. It is comprised of the following modules:

- **Web Interface** through which users are authenticated and where these users can select the rigs they wish to use. This is also the interface for administrators to manage their rigs and can provide a rig specific control interface (depending on the rig control implementation).
- **Scheduling server** which is responsible for queuing and assigning access to rigs. It queues rigs based on a specific rig identifier: a specific rig's name, a rig's type (so that multiples of the same equipment can be treated as a group) or *capabilities* (a tagging system to allow identification based on behavioural or configuration identifiers). The scheduling server also provides a registration interface to allow rigs to register themselves and declare their readiness to have users assigned to them. Determining this "'readiness"' is the domain of the Rig Client and will be described in this document in terms of the Rig Client out-of-session testing interface).

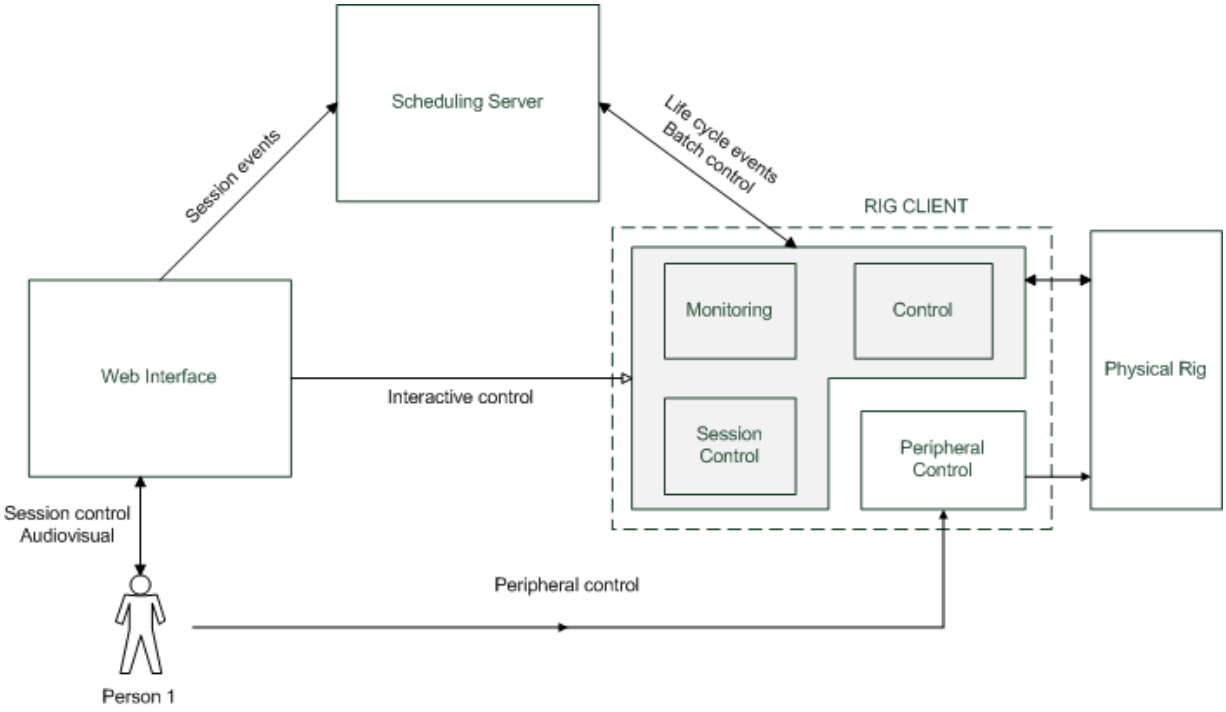

Figure 2.1: Sahara architecture overview

- **The Rig Client** is the intermediary between the Scheduling server and the physical rig. The Rig Client is explained in detail in this document.

2.4 Rig Client

The Rig Client (RC) is a software component that provides a software abstraction of a rig. It is the intermediary between a component that decides whom may have access to the rigs (the scheduling server) and its physical hardware (the rig itself) which provides:

- Session control - the mechanism of allowing externally determined users, master access to its rig (a mandatory function). The time between granting access to the master user and revoking access from a master user is termed a 'session'.
- Rig monitoring – testing of rig devices to ensure rig integrity (a mandatory function)
- Rig control - access to rig devices including control from uploaded instructions and/or direct control (an optional function). Direct control is named 'primitive' control because it instructs rig devices in terms of low level operations (for example, voltages, degrees or bit register values).

The main operations of the Rig Client are to:

- allow and revoke access to a rig from master and slave users (session control);

- provide notification to the rig user(s) of an externally provided notification message;
- allow testing of a rig to determine its status as available and usable.
- provide rig specific information to a user, provided the user has been assigned access;
- provide rig control from an uploaded instruction file, provided the user has been assigned master or active slave access;
- provide the ability to do synchronous rig control, provided the user has been assigned master access or sufficient slave access.

2.5 Web Interface

The Web Interface is the user interface for Sahara.

The WI has been designed to allow each institution to customise it as required. It provides for the ability for each deployment to override certain pages and features with institution specific content. Existing default content is used if no overrides are provided.

The mandatory development parts are the authentication of the user and the rig specific control interfaces.

In Sahara 2.0 the following functionality and features are included in the WI component of Sahara:

- Rig Type Page – this page is rig specific. There are 'widgets', termed 'elements', provided to include common function components onto the rig type page (for example, an applet to launch seamless remote desktop applications), but this should be designed and implemented according to the requirements of the rig type.
- Customisable common header – default Sahara headers and footers are supplied with the ability to customise these to the institution graphics. Feedback form – ability to send feedback, or fault information on Sahara.
- Contextual help dialogs – rendered place holders where information about specific pages may be provided.
- Login page – where users supply their login user name and password. This function can be implemented to interface to the institution identity providers and/or "local" user authorisation can be implemented. A pluggable authentication system is provided to chain multiple authentication sources together, with associated pre-access user setup.
- Rig Selection Page – page displaying all the rig types and specific rigs that a user has permission to queue for. It indicates which permissions are currently valid, which permissions have a lock associated with them and which are free to be queued for.

- Laboratory Rigs page – page for information on the rigs available. The default Sahara page contains images of the University of Technology, Sydney Remote Laboratory rigs.
- Demonstrations page – still under development, this page will allow unauthenticated access to the rigs to be used for demonstrations with appropriate configuration to set the conditions of demonstration access.
- News page – this page by default contains news on what has been happening with Sahara and Labshare. It can be used for site specific information.
- FAQ page – this page by default has frequently asked questions about remote laboratories. This page can be used to provide site information in a question, answer format.
- Contact Us page – this page by default has the contact details for Sahara Technical Development team. It should contain details of local support contacts.
- Administration and reporting functionality – under development.

2.6 Scheduling Server

Similar decription of scheduling server, why it must not be modified etc

Chapter 3

Development Approach

a discussion of how to to approach development for Sahara, figuring out what you need to develop, howit is structured etc. Below is the existing content that may go here, work to be done on putting the right chapters in the correct sections and adding latest revioew comments. Rig CLient struceture - here or previous chapter?

3.1 Purpose

The rig client translates the abstract concept of rig operations as requested by the scheduling server into specific rig behaviour. The development of the rig client specifies what constitutes this behaviour.

3.2 Requirements

In order to start developing the rig client the following design decisions should be made about the rig's design:

1. **How is singular access granted to a user?**

   There must be a mechanism whereby a user can be given access to a rig and no other (unauthorised) user can access the rig during this time.

   For example: a rig can be accessed using a remote desktop as a proxy to the control interface. In this case, the remote desktop allows the user to be "remote" and by adding a single user to the 'Remote Desktop Users' group access is granted to the control application. This group is used by Windows Terminal Services to grant access to Remote Desktops. As a restriction is set to prohibit other users login access, only the assigned user can use the rig.

2. **How is the rig controlled, and what constraints are there?**

   If the rig already exists and is already computer controlled, there will not be a need to duplicate control using the rig client.

If the rig does not yet exists consider the following:

- Will the rig be interactive or can it be used without the user being present (eg controlled by uploading an instruction file)?
- Will an interactive rig have its own implemented control interface? This is often suitable when there is a standard industry control interface (e.g. off-the-shelf PLC controllers) or an easily implemented application (e.g. LabView application).
- If there is no available control interface, does the rig have an applicable API for use with Java (i.e. Java, JVM hosted or JNI capable)? This would allow direct control using the Rig Client if suitable for the rig type. NOTE: Direct control is not suitable for rigs that require high fidelity (i.e. consistent timing and rapid responses) as it is implemented using HTTP requests and the timing is variable and cannot be guaranteed.

3. **What tests will be used to determine whether the rig is in a usable state?**

   It is desired that the rig determine its own status as being ready to assign to a user. This requires decisions on what constitutes being 'ready for use' for the specific rig. Tests to determine this readiness can include, for example:

   - Existence (network, device nodes) - can the device be contacted.
   - Environment (power, air pressure) - does the current operating environment support the needs of the rig.
   - Hardware (pistons, actuators, solenoids, motors) - can the hardware be commanded to be perform its desired actions.
   - Sensor (LVDS, magnetostrictive, temperature, time) - does a change in the hardware's state generate the desired measurable result with the tolerance of the sensors

3.2.1 Additional (optional) functions that can be considered

1. **What actions (if any) need to be performed to reset the rig?**

   If it is important that the rig be in a known state before the next session starts, or if leaving it in certain states is damaging to the rig, this needs to be included.

2. **What requirements are there to allow users to be notified of status/update messages while controlling the rig?**

   Messages concerning the status of a session or any updates the user needs should be directed to the rig control interface as that is where the users' attention will be. This may require a notification implementation (eg messages to be sent to a Windows or UNIX console session to generate an operating system notification dialogue).

3. **What method (if any) will be used to determine whether a user is actively using the session they have been granted?**

   This is used, for example, to end sessions of idle users. An example of this would be to determine whether or not the user has actually logged into the Remote Desktop console for the rig they have a session for.

4. **How (if at all) slave access will be granted to users?**

   Does the activity being designed for require slave users to be granted access, and if so how will this be done? (NOTE: The collaboration model between multiple users is a master – slave model. The master user is the session initiator and the user who can terminate a session. Slave users can be assigned access to a session to either passively view the session or actively participate in rig control.)

5. **How can activity or use be detected from the rig?**

To free rigs it that are assigned but not actively being used, the laboratory software needs to know whether the rig is actually being used. It is not possible to reliably determine this without knowing the context of the rigs so it is up to rig author to implement activity detection (with no specific activity detection implementation, the assumption is there is always activity). Some options to detect this are:

- Windows or UNIX console session list
- Some explicit control to turn on use, which may timeout.

3.3 Web Interface Development Approach

taken from original other section - must be modified

The Sahara approach to a customisable interface allows institute specific partial scripts to be written that can customise an institution's site, and the rig type pages.

The partial scripts are created in an appropriate directory with the name of the institution (which must match the configured institution in the config.ini file. For UTS, this directory would be: INSTALL_DIR/institution/¡configured_institution¿/scripts/

The Sahara Web Interface implementation first checks in this directory for any scripts and will use them if they exist, otherwise will use the default scripts. The mandatory rule when writing new code for the web interface is to put institution specific code into the institution directory, and use the public directory for code and images that are made available to all:

- CSS files – in INSTALL_DIR/public/css
- Images – in INSTALL_DIR/public/images

- Javascript – in INSTALL_DIR/js/...

  move to "'tips"' Note: In referencing the files that the web browser must load, use is made of the baseUrl() function. The function baseUrl() will return the preceding virtual directory of URLs, allowing the correct relative path of resources to be set for the browser to load (even if the site is in a virtual directory and not the document root). For example, to call a CSS file:
  $this-¿baseUrl('css/info.css') baseUrl() is a view helper and must be called from a Zend_View instance.

3.4 Rig Client Development Approach

Once the design considerations above have been defined, the development approach for the rig client (according to this document) can be determined. Based on the results from the questions above, the rigs can be categorised into one of three main categories of rig clients.

Peripheral control rig: This is a rig that is controlled only by another control application. In this case, all that needs to be defined are the actions (classes implementing a specific 'IAction' interface) that will be used for access, notification, testing, reset and activity detection.

Rig with primitive (synchronous) control: This is a rig that includes control directly to the rig through the Rig Client. This can be control directly to the hardware (eg turning on ports on a LabJack) or as an intermediary between the WI and the program that communicates directly with the hardware (eg interfacing between the WI and a LabView web service). In this case, the "translation" of user commands to the physical rig must be implemented. This is done in addition to defining the actions required for access, notification, testing, reset and activity detection.

Rig with batch mode control: This is a rig that includes control of the rig by an uploaded file which is not bound to the users presence (i.e. can be run later as a batch job after the user has logged out). In this case, the developer needs to specify how the file is verified, what command is invoked on the uploaded file, what actions to perform before and after executing the file etc. This is done in addition to defining the actions required for access, notification, testing, reset and activity detection.

For all development categories, the library of existing actions should be investigated to see if it holds actions that are required. If the actions already exist (e.g. access to a remote desktop, testing network connection) they can be used by the developer.

3.5 Rig Client Structure

Every new rig is described by a deployment of the rig client. At its most basic, this provides an interface to the rig and allows the rigs to provide the information it requires to the scheduling server (eg name, rig type, rig capabilities), for sessions to be started, stopped and monitored, and for the rig to be "exercised". These functions are described

by the IRig interface. Each rig type requires its own rig client which specifies the actions associated with the particular rig type. For most types of rigs this can be done through the use of a derivation of the AbstractRig class which implements the IRig interface.

The Sahara rig client development has favoured the use of re-usable classes which are collated into the rig client commons project. The AbstractRig class enables this by creating an rig class that needs only the users action classes to be written and specified. The tasks of registering actions, "executing" actions and implementing the IRig interface, are done. Please see Figure 2 below for the structure of the classes.

A number of different rig type classes exist which extend this AbstractRig:

- AbstractControlledRig (an abstract rig type class) – also implements the rig client control interface.
- ConfiguredRig (a concrete rig type class) – uses configuration to decide which classes to use
- ConfiguredControlledRig (a concrete rig type class) – also implements the rig client control interface and uses configuration to decide which classes to use

The AbstractRig class contains (among other things) a list of the actions for that rig type under the following categories:

- Access actions (IaccessAction)
- Slave Access Actions (ISlaveAccess)
- Reset actions (IResetAction)
- Notification actions (INotifyAction)
- Test actions (ITestAction)
- Activity detection actions (IActivityDetectorAction)

The rig type classes and actions are described in the later in this document.

3.6 Rig Type Classes

There are three rig type classes that have been written to assist in implementing the rig client. Each of these include functionality and methods for a more specific type of rig client than the AbstractRig alone. These correspond to the different rig types categories described in the chapter on Development Approach. Please see figure below for the class diagram showing the relationship between the abstract and concrete classes.

change and add figure Figure 2: Rig Type Class Diagram

3.6.1 AbstractControlledRig

The AbstractControlledRig is used to for rigs which require primitive control. In this case, the rig can be directly controlled using the rig client. The AbstractControlledRig class sets up the primitive Front Controller for the rig type to route primitive control requests. This rig type class also includes batch control. (Primitive control functionality is described below). This is an abstract class that requires additional implementation to be used for a Rig Client

3.6.2 ConfiguredRig

The ConfiguredRig is used to for rigs which have peripheral control. This rig type class allows the user to specify which actions to load. It is most suitable when all the functionality can easily be described in actions and only peripheral control is used. (The example below for the Loaded Beam is a ConfiguredRig). This is a concrete class that implements all the behaviour required of its underlying abstract class. It is ready for use as a Rig Client with the necessary configuration complete.

3.6.3 ConfiguredControlledRig

The ConfiguredControlledRig is used to for rigs which require batch mode as well as having peripheral control. This rig uses a "configured batch runner" i.e. a configuration file is used at run time to specify what to run, what arguments to provide and what environment variables to set. This is a concrete class that implements all the behaviour required of its underlying abstract class. It is ready for use as a Rig Client with the necessary configuration complete.

If none of these rig type classes are suitable for the rig, a new rig type class can be created that extends these, the abstract classes and implements the interfaces as required.

3.6.4 Rig Client Class Loading

In order to load the correct rig type class, this must be made visible to the Rig Client and be configured in the rigclient.properties file.

Place the code containing the jar file in the INSTALL_DIR/lib/ directory (create the directory if it does not exist). This directory contents are iterated and any jar files found are automatically added to the rig client classpath when the Rig Client starts. To verify the jar files are correctly added to class path, consult the rigclientservice.log file in the installation directory of the rig client. This will show each of the jar files that have been added to classpath.

If desired, explicit jar file paths can be configured using the INSTALL_DIR/config/rigclient_service.ini file. The Extra_Lib property contains a semi-colon separated list of jar files to add to class path. This may be useful to allow... *complete here*

The property Rig_Class should contain the fully qualified class name of the rig type class i.e. either an already configured class (eg au.edu.uts.eng.remotelabs.rigclient.rig.ConfiguredRig), or one that has been written for the rig type and is in a jar file on classpath.

3.7 Action Types

When a method is called on a rig type that extends the AbstractRig, the action list created for the corresponding category is iterated through and all the listed actions are initiated. For example, when a session ends and the tests action need to be executed, the startTest method will be called which will execute in order all the test actions that have previosuly been registered by calling the ITestAction startTest method. list testActions (a list of ITestAction instances) and call the method startTest on each action listed.

insert figure Figure 3: Rig Client action interfaces

To include a specific action in the list, they must be "registered" using the registerAction method.

If multiple actions are registered for an action type, these actions are always run sequentially. For Test, Notify, Activity Detection and Reset actions the actions are run in the order that they are registered with the Rig Client. For Access actions, they are run in the order they are registered when assigning access to the rig, and in the reverse order when revoking access from the rig. This is to allow correct sequencing for actions that have dependencies.

To create an action implementation, the associated interface class (eg IAccessAction or ITestAction) must be implemented. In the future, there will be many cases where there will be existing actions that have been written for existing rigs. These action classes can be re-used and the actions simply registered in the action lists described above.

Below are some details and specific examples of the different types of actions.

3.7.1 Access Actions

NOTE: In writing the access actions, keep in mind that in Sahara 2.0, the rig client allocation communication is synchronous and the rig client waits for the results of all the access action. For this reason, access actions should complete quickly. Any deadlocking of an access action will deadlock the rig client. In future releases, this will be changed to provide the option to use callback allocation communication.

The access actions of assign and revoke must be written into a class that implements the IAccessAction interface. Attention should be paid to the order that access actions need to be executed: these actions are run in the order they are registered with the rig client when assigning access to the rig, and in the reverse order when revoking access from the rig.

See the example for RemoteDesktopAccessAction.

3.7.2 Test Actions

NOTE: Test actions should run in their own threads.

For test actions, the methods startTest and stopTest must be written in a class that implements the runnable interface ITestAction. In addition, the methods getStatus, getReason and setInterval must be implemented. For this, the class AbstractTestAction has been written which uses a run flag to implement ITestAction.run and set up a persistent daemon thread that exists for the life of the rig client. This class also introduces a number of flags that can be used to tune test execution periodicity:

- isPeriodic - Specifies if the test interval is periodic or if it runs at a random interval.. Generally tests which result in visible effects (i.e. something moves) should not be periodic to mimic real lab use (not everything lab runs at the same time). The default is for tests to be periodic.
- isSetIntervalHonoured - Specifies if the test interval is honoured if set to a different value. The default is to honour the test interval.
- doLightDarkSchedule - Specifies if the test run interval is reduced at night time.

A detailed description of the implementation of the Ping Test is given below; the principles apply to the other tests as well.

3.7.3 Notify Actions

The INotifyAction interface must be implemented and the notify(message, users) method implemented in order to produce messages where the users focus is, for example on the users Remote Desktop session for Remote Desktop type experiments.

See example below.

3.7.4 Activity Detection Actions

IActivityDetectionAction interface must be implemented and the detectActivity() method written. For example, this can be done with RDPActivityDetectorAction.

3.7.5 Reset Actions

IResetAction interface must be implemented and the reset() method coded. For example, this reset action could power down the rig, then power it back up, to return the rig to a known, safe state.

3.8 Rig States

During operation, the Rig Client can be in various states that describe its life cycle. Please see the diagram below for a description of these states and transition.

update and include diagram Figure 4: Rig Client state diagram

Chapter 4

Development Environment

Suggestions on using Eclipse, how to set up projects, how to use ant etc

Chapter 5

Fully Peripherally Controlled Rig

a discussion of the peripherally controlled rig. Below is the existing content that will be restructured (along with other content) into the following sections

5.1 Theory of Operation

A fully peripherally controlled rig is one that is controlled by an application external to Sahara. In this case, the Rig Client is still responsible for access, revocation, notifying users of messages, detecting user activity and determining and reporting on the rig status, but there is no requirement to control the rig while it is in session.

5.2 Specific Implementation

5.3 Case Study

The following description of a fully peripherally controlled rig client development will be done using the example of a (real) Loaded Beam rig. The Loaded Beam rig applies forces to a beam and measures the beams deformation. It is controlled using a Labview interface that is accessible by remote desktop to the machine hosting the controller.

There are two ways to implement this example. The first would be to create a specific implementation eg a LoadedBeamRig class. In this case, the actions are determined at compile time instead of run time. The alternative is using the ConfiguredRig class which allows the actions to be entered into the properties file. In this case, actions are determined only at run time but it is more suitable for rapid prototyping and development. The specifics for the Loaded beam rig are described here:

- Rig Type
  The Loaded Beam is an example of a rig that is fully controlled by a peripheral control program (written in LabVIEW as a desktop application). In this case, either a specific implementation extending AbstractControlledRig, or the ConfiguredRig

type must be set up which configures the actions used by the rig client to perform its functions.

- Access Actions
  The Loaded Beam experiment is accessed using a Remote Desktop connection to a machine on which the LabVIEW control program runs. The Rig Client will run on the same machine, and access is granted to the Remote Desktop by adding the users' name to a "Remote Desktop Users" group. Access is revoked by removing the user from the group so that they cannot get access to the Remote Desktop.

- Test Actions
  In terms of testing, the Loaded Beam can be tested using a "ping test" to see that it is still on-line and also apply a voltage and read the results to see that it is functioning as required. Additionally, the camera is checked to determine whether this is functioning as required

- Notification
  Notices must be sent to the Windows Remote Desktop, eg to notify of an impending log-off.

- Reset
  The Loaded Beam must be reset to the state where there is no load on the beam to prevent permanent deformation.

- Activity Detection
  The Loaded Beam is said to have "activity" if the Remote Desktop is running.

The AbstractRig class implements the IRig interface and in most cases will be appropriate to be used to for a specific rig client. For the Loaded Beam, a class is created that extends AbstractRig and registers each of the actions used by the Loaded Beam rig.

5.3.1 LoadedBeamRig Class

For our example, the LoadedBeamRig class (which extend AbstractControlledRig) includes the code necessary to register the actions for the Loaded Beam. For this, an instance of each action is created, and this is then "registered" enabling the Rig Client to perform the required rig specific action. This is done in the init() method. For example, for the Loaded Beam actions is:

update with latest

```
protected void init()
{ /* ----------------------------------------------------------------------
* 1) Access actions.  --
* ----------------------------------------------------------------------*/
this.registerAction(new RemoteDesktopAccessAction(), ActionType.ACCESS);

/* ----------------------------------------------------------------------
* 2) Slave access actions.  --
* ----------------------------------------------------------------------*/
/* None currently.  */

/* ----------------------------------------------------------------------
* 3) Reset actions.  --
* ----------------------------------------------------------------------*/
this.registerAction(new DeleteFilesResetAction(), ActionType.RESET);

/* ----------------------------------------------------------------------
* 4) Notify actions.  --
* ----------------------------------------------------------------------*/
this.registerAction(new WindowsMsgNotifyAction(), ActionType.NOTIFY);

/* ----------------------------------------------------------------------
* 5) Test actions.  --
* ----------------------------------------------------------------------*/

/* ----------------------------------------------------------------------
* -- 6) Detection actions.  --
* ----------------------------------------------------------------------*/
this.registerAction(new RDPActivityDetectorAction(), ActionType.DETECT);
}
```

Then, during operation, when the assign method is called for this rig, the access action registered is used to find which assign implementation to use. The rig client will iterate through list of ACCESS type actions registered, and perform the assign method for each one.

The same process occurs with action types.

5.3.2 Configuration

For our implementation, the rigclient.properties configuration values that need to be set are:

- **Rig_Name** – this is a unique identifier for the rig. In our case "Loaded_Beam_1"
- **Rig_Type** – identifies the rig as a member of this type (a group of fungible rigs). In our case "LoadedBeam"
- **Rig_Capabilities** – tags that "describe" the rig. In our case "beam,civil, deflection, cantilever, Young's Modulus, compression, tension"
- **Rig_Class** – the qualified class of name of the rig type class implementing the rig. In our case: au.edu.uts.remotelabs.loadedbeam.LoadedBeamRig

5.3.3 Alternative Implementation

An alternative way to implement this would be to use the ConfiguredRig class for the Loaded Beam. In this case, instead of adding and registering the actions as above in the rig type class, the actions are configured in the rigclient.properties file which is then read to determine which actions to register.

In this case the Rig_Type_Class property would be set to the ConfiguredRig class (au.edu.uts.remotela and the following additional properties would need to be configured:

- **Action_Package_Prefixes** – the package name where the action interface classes can be found (used if the fully qualified action class names fail)
- **Access_Actions** – list of access actions to load (eg access.RemoteDesktopAccessAction)
- **Slave_Access_Actions** – list of slave actions to load
- **Notify_Actions** – list of notification actions to load
- **Detection_Actions** – list of activity detection actions to load
- **Reset_Actions** – list of reset actions to load
- **Test_Actions** – list of test actions to load

NOTE: In order to resolve the correct class from the configured value:

1. The configured value is checked to see if exists as a class name. If it does, that class is used.
2. Each of the package prefixes configured in 'Action_Package_Prefixes' is prepended to the configured class name and that is checked to see if it that exists as a class name. If it does, that class is used.

5.3.4 Actions Examples

Access Action – RemoteDesktopAccessAction

This access action adds and removes users from the Remote Desktop Users group which controls who may remotely login to a windows console using RDP. The action only works for Windows and will fail on any other platforms.

Access is granted using the net localgroup command to add users to a user group:

- **net localgroup group** – used to query who belong to the group
- **net localgroup group [domain/]username /ADD** – used to add user username from (optional domain domain) to the user group group
- **net localgroup [group domain/]username /DELETE** – used to remove user username from (optional domain domain) from the user group group

The interface methods inherited from IAccessAction are:

- **assign(String name)** – an implementation to assign access to user name. In the RemoteDesktopAccessAction, this first checks whether a user is part of the configured group. If not, the user is added and the result verified. The verification is done by checking the exit code of the windows command.
- **revoke(String name)** – an implementation to revoke access from user name. In this case, the user's session is terminated. This is to ensure that a user may only use the rig between the time they are assigned access and the time that access is revoked. The user's sessions are detected using qwinsta, and stopped using logoff. The command to delete the user from the group is then executed and the results verified.
- **getActionType()** – returns the action type, in this case "access"
- **getFailureReason()** – an implementation specific reason for failure. Returns null if the action succeeded

Configuration For each action implemented, the developer can edit the rigclient.properties file (or create a new one) that includes configurable parameters. In the case of the Remote Desktop Access action the following parameters were added to the rigclient.properties file:

- Remote_Desktop_Windows_Domain – the windows domain that the user belongs to (optional to be supplied)

Test Action – PingTestAction

For the Ping Test, the AbstractTestAction is extended and the implementation pings the host to determine if it is up. The Ping Test is implemented in the PingTestAction class. This test action pings the host to see if it is up. It has been implemented with a configurable test interval (default test interval of 30 seconds) and is designed to run periodically. For the ping test, the time interval set using setInterval method is not honoured.
The methods that must be implemented from IAccessAction are:

- **getStatus()** – calls to this method must reflect the status of the rig. Returns true if tests pass, false if tests have failed.
- **getReason()** – the reason that the test has failed. Returns null if the test has passed.
- **setInterval(int interval)** – sets how often the test is executed in minutes (can be overwritten with flag isSetIntervalHonoured if the abstract class AbtractTestAction is extended instead of directly implementing ITestAction.)

In addition to this, the AbstractTestAction class required the following methods to be implemented:

- **setUp()** – method that runs before any tests are executed. In the Ping Test case this reads the properties from the file to collect the osts to ping and the arguments to use.
- **doTest()** – the actual test method. This is invoked at each test interval when the rig client is running in test mode. If this method is long in duration (more than a few seconds), it should regularly check whether it has been instructed to stop or not (using the super runTest flag from AbstractTestAction).
- **tearDown()** – Cleanup method that is called when the test is shutdown.

Configuration For each action implemented, the developer can edit the rigclient.properties file (or create a new one) that includes configurable parameters. In the case of the Loaded Beam ping test action the following parameters were added to the rigclient.properties file:

- **Ping_Test_Host_¡n¿** - The host names for the ping test to verify for response, where 'n' is from 1 to the nth host to ping test, in order. The 'Ping_Test_Host_1' property is mandatory and each subsequent property is optional.
- **Ping_Test_Command** - The command to execute a ping. This is optional, with the default being 'ping' which is expected to be in $PATH in UNIX and %PATH% in Windows.
- **Ping_Test_Args** - The arguments to supply to the ping command. Ideally this should cause the ping command to ping the host once and have a timeout of a few seconds. The host address is always the last argument.
- **Ping_Test_Interval** - The time between ping tests in seconds.
- **Ping_Test_Fail_Threshold** - The number of times ping must fail before the ping test fails.

Notify Action – WindowsMsgNotifyAction

This example would uses msg command line executable and the specified message is sent to each user described in the method parameters. This is implemented in the WindowsMsgNotifyAction class. There are no configurable properties for this class.

Activity Detection Action – RDPActivityDetectionAction

change to latest For example, this can be done with RDPActivityDetectorAction class for Remote Desktop sessions. This class checks to see if the Remote Desktop is still in session using the qwinsta executable. Any existing session is used as validation for success – there is no check that the session corresponds to any user. There are no configurable properties for this class.

5.4 Hints and Tips

Chapter 6

Rig with Primitive Control

a discussion of the primitive controlled rig. Below is the existing content that will be restructured (along with other content) into the following sections

6.1 Theory of Operation

6.2 Specific Implementation

6.3 Case Study

6.4 Hints and Tips

The objective of primitive control is to bridge a request from the Web Interface to a specific action on the Rig Client that the user is assigned to. A rig with primitive control works through the rig client itself issuing control instructions to the rig. This is done by setting up an interface that uses the WI Primitive controller to map the url to a class and method with any number of parameters on the Rig Client.

By default, all session users can perform primitive control actions, that is master users, slave active users and slave passive users (where the master is the user initially given access to the rig, and the slave users are other users subsequently granted access by the master user to control and/or view the rig session). In order to restrict the use of primitive controls to certain users, filtering of role access may be implemented.

The following description details how to set up a primitively controlled rig and gives examples of how to develop primitive control actions. Please note that primitive control actions can be used with other rig types to perform required actions on the rig.

Configuration For our implementation, the rigclient.properties configuration values that need to be set are: Rig_Name – this is a unique identifier for the rig. In our case "FPGA_1" Rig_Type – identifies the rig as a member of this type (a group of fungible rigs), eg "FPGA" Rig_Capabilities – tokens that "describe" the rig, eg "FPGA, programming"

Rig_Type_Class – the qualified class of name of the rig type class implementing the rig. In our case: au.edu.uts.remotelabs.AbstractControlledRig. Action_Package_Prefixes – the package name where the action interface classes can be found (used if the fully qualified action class names fail) Access_Actions – list of access actions to load Slave_Access_Actions – list of slave actions to load Notify_Actions – list of notification actions to load Detection_Actions – list of detection actions to load Reset_Actions – list of reset actions to load Test_Actions – list of test actions to load

If the required actions do not exist, new action classes must be written that implement the specified action interfaces, as for the examples in Chapter 3.1.4. This works the same as for a peripherally controlled rig. Additionally, for primitive control, commands can be executed on the rig using the Sahara Web Interface. This is done using a web url that is configured to send a specific command with specified parameters to the rig. The implementation is described below. Primitive Control There are two parts to the implementation of primitive control: the part of the user interface that is updated with the information provided by the primitive control request (either by user interaction, or automatically), and the rig client implementation which includes the specific method invocation that the Web Interface request is bridged to. This is accomplished by setting up an interface that uses the Web Interfaces primitive control prefix (described below) to map a named controller and action (WI) to a specific class and method (Rig Client). Web Interface implementation The primitive control URL has the following format: http://webserver_IP_OR_hostname/ primitive/¡PrimitiveControlAction¿/ pc/ ¡PrimitiveController¿/pa/ ¡primitiveActionName¿/ ¡parameterKey¿/ ¡parameterValue¿/ ¡parameterKey¿ / ¡parameterValue¿ /...

Where: webserver_IP_OR_hostname is the hostname for the Web Interface primitive and ¡primitiveControlAction¿: indicate that this is to be a primitive control call with the specified action. The action bridges a primitive call to the in-session Rig Client. The actions specify what is done with the response parameter (if there is any) received from the rig client. The actions available for the primitive controller are file, echo and json (described below). This is a mandatory parameter. pc (or primitiveController) and ¡PrimitiveController¿: indicate for the name of the primitive controller that will be used. The value PrimitiveController will be the name of the java class that contains the action to be performed. This is a mandatory parameter. pa (or primitiveAction) and ¡primitiveActionName¿: indicate the name of the action to be run on the controller. The value primitiveActionName will be the name of the java method (without the suffix Action) that must be executed to perform this primitive control action. This is a mandatory parameter. ¡parameterKey¿ and ¡parameterValue¿: These are key-value pairs that indicate the parameter names and values for the primitive control action. There are some parameters associated with the primitiveControlAction that was specified (described below) , all other parameters indicate the parameters that should be passed when executing the specified method. These are optional parameters.

The three primitive control action types are: 1. file: This action indicates that, if a response parameter is specified, its value is a file to download. If no response parameter is specified, all the response parameters are returned as a file in the format name=value, name=value, ... etc. The file may be text or binary in nature, and (with a suitable mime-

type) may directly display on a web browser instead of a file download (this is an option to force file downloads). For example, a JPEG image file may be displayed onto the web browser provided the mime is set to 'image/jpeg'. There is a soft limitation of file size, due to discreet nature of communication. The file should not be larger then a couple of megabytes and should generally be kept in the low tens of kilobyte range. The file action should be used when the information requested from the Rig Client is a file, for example a results file to be made available to download. The possible parameters for the file action are: rp or responseParameter: the name of the response parameter mime: the mime type of the returned file. The mime type and subtype of the file should be separated with a '-' eg text/xml should be text-xml. fn or filename: name of the file. This forces a file download. tf or transform: any transformation that needs to be applied to the file. The currently implemented option for this value is: base64 – this interprets the return value as base64 and transforms it as required. To be used if the downloaded file is binary data. 2. echo: This action indicates that, if a response parameter is specified, its value is a echoed back. If no response parameter is specified, all the response parameters are returned in the format format name=value, name=value, ... etc. The echo is suitable for use with text type data that is to be directly provided to the browser without an interpretation. For example, this can be used with XML data for client side DOM parsing. The echo action is best used with a nominated response field that will return just the text content. The only parameter for the echo action is: rp or responseParameter: the name of the response parameter 3. json: This action indicates that that the response to the action is a JSON string. This action can be used when either of the other two is not suitable and the result is information from the rig client that needs to be manipulated or parsed further before being useful to the interface. It is most appropriate with action methods that provide map information to be obtained with AJAX calls. This is due the native interaction between JSON and Javascript that means the developer does not have to serialise and interpret XML or text, rather a Javascript object (or array of objects) is received with map keys that are callable as object properties.

Rig Client Implementation The Rig Client that uses primitive control must implement the primitive control interface IPrimitiveControl. The method declaration must match the specified format. The Rig Client processes the primitive control request using the "Front Controller" pattern to call the method primivitveAction on a class PrimitiveController (after checking whether the requesting user is assigned and their role satisfies any role filtering constraints, if implemented). The front controller (PrimitiveFront.java) is responsible for determining whether the request is valid in terms of there being an existing class and method matching the request. After this it is the responsibility of the implemented primitive controller to check for errors. A primitive controller class must implement the IPrimitiveController interface which specifies that the following methods should be implemented: initController - Method that is called during instantiation of the controller. preRoute - Method that is called before routing to an action of this class. postRoute - Method that is called after the completion of the routed action. This method is guaranteed to be called if preRoute successfully completes. cleanup - Method that is called when the controller is cleanup up (generally at the termination of sessions). Primitive controllers are "cached" so that if the controller has instance exists already, there is no new instance created. This allows for the class to be resolved and instantiated only once

during the session, instead of repeatedly. It also means that a state can be stored that will be persisted for all subsequent action method invocations, therefore allowing "stateful" control to be developed. Controller classes have the following life-cycle: 1. On first request of a controller instance, the controller is instantiated and added to the controller cache. 2. On the next and subsequent requests of a controller instance, the controller instance is recalled and reused to run action. 3. On termination of the masters rig session, clean up is called and the controller instance is discarded. It is safe to have resources (open files, handles...) as instance fields provided they are cleaned in the cleanup method. In order for the Front Controller to correctly route the primitive request to the correct action, the declaration for action methods implemented in the primitive controller must be in the form: public PrimitiveResponse primitiveActionNameAction(PrimitiveRequest) where primitiveActionName corresponds to the action specified in the primitive control request from the Web Interface. In the methods, the provided PrimitiveRequest argument contains a hash table of the request parameters to the action method. If the method cannot be resolved because it is not suitable declared, or throws an exception, the primitive control request will fail with the following error codes: 0: No error -1: Illegal controller or action argument. -2: Controller class not found. -3: Action method not found. -4: Security exception accessing action. -5: Illegal access to the action (the action method is not public). -6: Invalid action signature (does not take, and only take a PrimitiveRequest parameter). -7: Action has thrown an exception. -8: The preRoute method failed. -9: The ACL check does not allow the users role to perform the action. By convention, 0 should be used for no error, negative error codes should be resserved for routing errors and positive error codes for action method errors. Action methods should not throw exceptions but should return error codes. An exception that is thrown by the method will result in error code -7. Role Filtering for Primitive Control Actions For Sahara v3 and above the role (e.g. master, slave active or slave passive) of users allowed to invoke an action method may be filtered. If implemented without any role filtering, all assigned user roles can perform any primitive control action. It is advised that consideration be given to this implementation and if control is to be restricted, then filtering is done. The role filtering is done implemented per controller written in the Rig Client. The filtering is based on an "action control list" model that specifies the actions certain roles are allowed to execute. It is included by implementing one of the following interfaces for the controller: MasterAllowed – this allows only the Master role to perform primitive control SlaveActiveAllowed – this allows the Master and Slave Active roles to perform primitive control. The Slave passive role is not allowed to perform any primitive control action RoleFilteredAcl – this implements access control lists to filter access to primitive control. Access is determined with the following hierarchy: Master – has implicit access to all controls Slave Active – has implicit access to all actions that Slave Passive role has access to, as well as any actions in the Slave Active control list Slave Passive – has access only to the actions in the Slave Passive control list. The lists are created by populating slaveActiveActions list with those actions the Slave Active role is allowed access to, and similarly with the slavePassiveActions List for the Slave Passive role. iPrimitiveAcl – this can be used if none of the above interfaces suit the restriction requirements. This allows the rig developer to specify exactly which actions each user is allowed access to. For example, in order to allow all Master and Slave Active users access to all controls and restrict Slave Passive users from any con-

trols, using our FileTransfer example, the declaration for the class would be: public class FileTransferController implements SlaveActiveAllowed Annotation based Controllers In Sahara v3 and above controllers may be specified using annotations... Example of Primitive Control The primitive control implementation will be described with the help of an example FileTransfer functionality. This is primitive control functionality because the Web Interface is executing a command on the rig: an interface is generated that is periodically updated with all the files in a specified directory on the Rig machine, that are then displayed to the user on the interface, available for download. Stateful control is used to discriminate between new files and previously listed files. This allows the web interface to highlight the newly detected files. Firstly, the interface portion of the implementation requires that the a primitive control request be generated periodically to supply details of the files available for transfer. For the FileTransferController example, the url to transfer a file build.xml which has mime type text/xml and requires no transforming will be: http://localhost/primitive/file/pc/FileTransferController/pa/textFile/filename/build.xml/mime/tex xml hostname = localhost primitive_control_action = file primitive_controller = FileTransferController primitive_action = textFile parameter_key = filename; parameter_value = build.xml parameter_key = mime; parameter_value = text-xml The mime parameter is used to indicate how the response file should be interpreted as it is a parameter for the primitive type file. The parameter filename is NOT a parameter for type file so will be passed to the method in the form of an object including this key-value pair. This url is a primitive call to execute the method textFile in the class FileTransferController and pass the parameter filename to the method.

For the second part of the implementation, the Rig Client needs to supply the information requested. The class FileTransferController has been written which implements IPrimitiveControl. This class is written to transfer all files in a configurable source directory to the Web Interface (for download by the user). The following methods have been included: initController – reads the configurable source directory sets up parameters for storing files and their types. preRoute – nothing to do, returns true. textFileAction(request) - transfers the contents of the file specified by the request parameter. This is used for text files. binaryFileAction(request) – transforms the contents of the file specified by request parameter. listFileAction(request) – lists the files in the configured source directory that matches the optional extension or regular expression provided to match the files. If the parameters is not provided matching isn't done. deleteFileAction(request) - deletes the file specified in the request parameter from the source directory. postRoute() – nothing to do, returns true. cleanup() - nothing to do, returns true.

Chapter 7

Rig with Integral Batch Mode Control

a discussion of the batch controlled rig. Below is the existing content that will be restructured (along with other content) into the following sections

7.1 Theory of Operation

7.2 Specific Implementation

7.3 Case Study

7.4 Hints and Tips

Description of what development is required for a Rig Client for a rig that can be executed in batch mode. Developer must implement a ConfiguredControlledRig class that registers the Actions required for the rig, as well as identifying the ConfiguredBatchRunner that is associated with the rig If the actions required do not yet exist, the new action classes must be written that implement the specified action interfaces The ConfiguredBatchRunner class must be created for this rig which specifies what is invoked, how the file is verified, how it executes etc

Including examples of class and method implementations from Remote Labs

Chapter 8

Rig Type Page Development

When a user is granted access to a rig, they are directed to a rig specific page that displays the interface for the rig, or instructions on how to access the rig. There is a standard template for the rig page which includes the:

- header
- footer
- rig name (from the Rig Client)
- a Session Timer panel including the amount of time the session has been active and a countdown of the remaining session time (according to the session duration configured per permission)
- "Feedback" button
- "Finish Session" button
- "Logout" button

Additionally, if no rig specific page exists, the default Remote Desktop rig page is used which includes:

- Remote Desktop Launch element
- A camera element centred in the page
- "Video Format" element allowing users to select the format the want (configured in the rig client properties file, see Rig Client Development Handbook)

The rig specific interface is generated using a "partial script" which is rendered in addition to the standard template.

The figure below indicates some of the optional and mandatory parameters on the rig page. Please note that the rig name is an optional parameter and the buttons on the left and right can be repositioned if required (eg all moved to the left hand side).

include figure Figure 5 - Screen shot of example rig type page

8.1 Rig Specific Elements

The Sahara rig page development has favoured the use of re-usable classes that render themselves, rather than explicit html for creating the rig pages. Users are free to implement their elements as they would like, but we provide a guide as to how it has been done by the Sahara team.

Please see Appendix C for a list of the elements that are currently available.

In order to create a rig element elementName, following the Sahara style, the following needs to be completed:

- Create a partial script INSTALL_DIR/library/Sahara/Session/Element/¡ElementName¿/¡_elem directory that includes the html for the element itself
- Create a PHP class for your element that extends Sahara_Session_Element: INSTALL_DIR/library/Sahara/Session/Element/¡ElementName¿.php. This class must implement the abstract method render() which should return the markup of the generated element to add to the web page. This is best implemented using the "view renderer" that each element has to render the partial view script of the element..
- If required, put CSS information in INSTALL_DIR/public/css/elements directory.
- If required, put any Javascript function in the INSTALL_DIR/public/js/elements directory.

8.2 Rig Specific Partial Script

To create a rig specific partial script the following convention must be applied:

- Create a directory INSTALL_DIR/institution/instition_name/scripts where institution_name is the name configured in the rig client properties file, eg C: Program Files/Sahara/WI/institution/UTS/scripts"
- Within the new directory create the file RigType.phtml where RigType is the rig type configured in the rig client properties file.

This partial script should have the rendering for what is to be displayed on the rig page and how this should be done to accommodate the rig interface Some options for this are, already created elements, such as:

- RDP Launch element which allows a user to click on a "Launch" button and gain access to a Remote Desktop Session
- This is configured in the rigclient.properties file by changing the following Remote_Desktop_Options property.

- Camera Panels – one or more camera panels can be configured to be displayed anywhere on the screen. This is configured in the rigclient.properties file by changing the following Number_of_Cameras and Camera_x (to specify formats) properties.

  Video Format Selector – part of the Camera element, allows the user to select the format they want to see the camera feed in from a range of configured values

- Session Files element – Allows for file transfer from a specified directory on the Rig Client machine to the users machine. It displays file that exist in a directory specified by the File_Transfer_Directory property in the rigclient.properties file.

- Left/Right Push - The left hand side and right hand side page content is, by default, set at a fixed width but there is code that allows these to be pushed closer to left and right screen edges.

Chapter 9

Site Customisation

a discussion of how to customise each site. Below is the existing content that will be restructured

The Web Interface has been developed in PHP using the Zend Framework. The Zend Framework implements a Model-View-Controller architecture. The following components of the Web Interface can be changed for Sahara installations.

- Rig Type Page – this page is rig specific. If it is not written, a default page is displayed for that rig type.
- Customisable site header
- Laboratory Rigs page
- News page
- FAQ page
- Contact Us page
- Configuration parameters (eg email addresses for feedback)

The Web Interface configuration file /config/config.ini has some configurable parameters for this.

9.1 Site Customisation

9.1.1 Header

The header and footer are included on each page of the Sahara Site. These can be customised using the configuration file or further if needed by changing the header partial script. The configuration parameters that can be are:

- header.logoGraphic – the institution graphic displayed on the top left

- header.logoLink – the link to follow when clicking on the institution logo.
- header.nameImage – the image to be displayed to the right of the institution logo

The header images need to fit into the following dimensions to prevent cropping or resizing: Height – 70px Header Image width – 580px

9.1.2 Feedback Form

The feedback form is overlayed on whichever page is shown when the Feedback button is pressed. The configurable parameters for the feedback are the email addresses to which the feedback should be sent eg

feedback.address[] = mdiponio@eng.uts.edu.au feedback.address[] = tmachet@eng.uts.edu.au

9.1.3 Laboratory Rig Page

update to new galleria There is a Laboratory Rig page which currently holds a gallery style display of images.

If a similar gallery style display is to be implemented, this can be done by creating the institute specific script INSTALL_DIR/institution/¡Institution_name¿/Labinfo.php, where institution_name matches that institution name configured in the config.ini file for the Web Interface. The class name of the script must be ¡Institution¿_¡Info_type¿.php (fin order for Zend's auto loading to work).

This script should have a list of arrays with each array containing the information for the image. The image information is an associative array containing:

filename =¿ image filename (from baseUrl)

alt =¿ subtitle for image

title =¿ title for image

For example: private $Images = array (array("filename" =¿ "uts/images/image1.jpg", "alt" =¿ "Description of Image 1", "title" =¿ "Title of image 1"), array("filename" =¿ "uts/images/image2.jpg", "alt" =¿ "Description of Image 2", "title" =¿ "Title of image 2"), array("filename" =¿ "uts/images/image3.jpg", "alt" =¿ "Description of Image 3", "title" =¿ "Title of image 3"));

If the gallery style is not wanted, this page can be replaced with whatever laboratory information is desired by overwriting the file INSTALL_DIR/application/views/scripts/labinfo/index.phtm

9.1.4 Information Pages

The "information" pages cover the Contacts Us page, News page and Frequently Asked Questions (FAQ) page.

Similarly to the Laboratory Information page, the other information pages can be written for the specific institute or the default can be used, or completely overwritten.

For creating an institute specific Contacts Us page, create the script INSTALL_DIR/institution/¡Insti
where institution_name matches that institution name configured in the config.ini file for the Web Interface.

This script should set up the information to be displayed on the Contact Us page in the table shown. The structure is an associative array which contains role =¿ contact information. The contact information is again an associative array with any key =¿ value pair that should be displayed (they will be displayed in a table as field =¿ value). The script should have a method getContacts() which returns this array.

Any links must have the html mark up included.

For example: private $contacts = array ("Operational" =¿ array("Contact Name:" =¿ "Michel de la Villefromoy", "Contact Phone:" =¿ "(02) 9514 2406", "Contact Address:" =¿ "UTS Building 1", "" =¿ "CB01.23.16", "Contact Email:" =¿ "¡a href=m̈ailto:name@address¿Ope "Technical" =¿ array("Contact Name:" =¿ "Tania Machet", "Contact Phone:" =¿ "(02) 9514 2975", "Contact Address:" =¿ "UTS Building 1", "" =¿ "CB01.23.16", "Contact Email:" =¿ "¡a href=m̈ailto:name@address¿̈Technical¡/a¿"));

For creating an institute specific FAQ page, create the script INSTALL_DIR/institution/¡Institution_n
where institution_name matches that institution name configured in the config.ini file for the Web Interface. This script should set up the information to be displayed on the FAQ page in the default accordion style. The structure is list of associative arrays which contain the key value pairs: question =¿ question text answer =¿ answer text Any links or editing must have the html mark up included. The script should have a method get-FAQ() which returns this array. For example: private $FAQ = array(array("question" =¿ "What is Sahara?", "answer" =¿ "Sahara is the software used to access remote laboratories anywhere, any time. Sahara is a suite of open source software components that has been developed at UTS under the LabShare program. ¡p¿ You are currently using Sahara V1.0. There will be further releases and updates available from our repository at our ¡a href=ḧttp://sourceforge.net/projects/labshare-sahara/¿̈ open source repository¡/a¿. ¡p¿ For more information or any questions regarding Sahara, please use the S̈end Feedbackẗab or the contact details provided."),

array("question" =¿ "So what is a remote laboratory anyway?", "answer" =¿ "A remote lab is a set of laboratory apparatus and equipment which is configured for remote usage over a network - usually the Internet. As much as possible, in setting up a remote laboratory, the goal should be to preserve the same apparatus and equipment (with the same limitations and imperfections) as would be used if the students were proximate to the equipment as per a conventional laboratory.")));

For creating an institute specific News page, create the script INSTALL_DIR/institution/¡Institution_
where institution_name matches that institution name configured in the config.ini file for the Web Interface. This script should set up the information to be displayed on the News page in the default accordion style. The structure is list of associative arrays which contain the key value pairs: header =¿ News item header text info =¿ News item information Any links or editing must have the html mark up included. The script should have a method getNews() which returns this array.