Diagnostics Integration Module Guidelines

So you have created a new integration module. That is awesome! What should we check before incorporating this module into the Diagnostics portal?

Contents

[Criteria 2](#_Toc509837786)

[Introduction 2](#_Toc509837787)

[Structure 2](#_Toc509837788)

[Example 2](#_Toc509837789)

[Code style 3](#_Toc509837790)

[Coding guidelines 3](#_Toc509837791)

[Testing 3](#_Toc509837792)

# Criteria

## Introduction

An integration module is a way to define Diagnostics custom functions through Magik code. These custom functions will trigger Diagnostics event creation each time a specified Magik method is run. The integration module is used as a container to group custom function definitions that apply to functionality defined in the same Smallworld module (core or customer module). Diagnostics only loads an integration module if the targeted Smallworld module is present in the image. This simplifies image composition and avoids builds errors. In order for this to work, the creator of the integration module must specify the name of the Smallworld module in which the targeted exemplars are defined. This module is called the integration target.

## Structure

An integration module is just another Smallworld module. It is defined in a folder and this folder contains a module.def, source, and resources.

Check the following things before submitting your integration module:

1. An integration module may define multiple custom functions, but all of these custom functions must target exemplars from the same Smallworld module. Check that the exemplars for which you have defined a custom function all reside in the same Smallworld module. You can do this by asking my\_target\_examplar.module\_name on the prompt.
2. The name of the folder in which the integration module is defined must match exactly the targeted module name. This is how Diagnostics can determine to which Smallworld module this integration module applies. \*
3. The module.def must specify a dependency on the diagnostics\_tracker module (version 1).
4. The name of the module (in module.def) must be unique. It should have the format diagnostics\_abc\_integration, where abc can be anything. Valid examples are: diagnostics\_merge\_integration and diagnostics\_roos\_sync\_manager\_src\_integration.
5. In your resources\base\data folder, place a copy of the dashboard XML your created in Splunk.

*\* From Diagnostics 1.5 onwards, it is no longer required that the folder name of your integration module matches your integration target. Instead, you can list your integration target as an additional dependency in your integration module's module.def. If you want to ensure backwards compatibility with Diagnostics 1.4, you should still define the integration target in the folder name.*

### Example

The following is just an example to clarify the above.

1. The custom function targets method database\_view.merge(). This is defined in the register.magik. The integration target is ds\_src (because database\_view.module\_name is ds\_src).
2. The name of the folder in which the integration module resides is ds\_src (because ds\_src is the integration target).
3. The module.def contains 1 required module: diagnostics\_tracker 1
4. The name of the integration module is diagnostics\_merge\_integration (in module.def).
5. There is a file called merge\_activity.xml in ds\_src\resources\base\data.

The complete ds\_src\module.def looks as follows:

diagnostics\_merge\_integration 1

description

Provides integration to generate Diagnostics events for Merge behaviour

end

requires

diagnostics\_tracker 1

end

## Code style

Please adhere to the following styling guidelines.

1. Above all your Magik files, include a header section where you specify the purpose, author, and creation date of the file.
2. Specify a package declaration in each Magik file.
3. Include a pragma for your defined methods.
4. Make sure your defined before and after methods do not already exist on the targeted exemplar. For example defining database\_view.diagnostics\_before\_merge() can be assumed to be safe, but database\_view.log() might already exist.
5. In your methods, include a comment stating the purpose, author, and creation date of the method.
6. Code should be self-explanatory. If not, add comments.
7. Do not include any (business) sensitive information in your code. After all your integration module will be shared :)

## Coding guidelines

The following should make sure that your code runs smoothly.

1. DO NOT CALL ANY METHODS THAT CHANGE THE STATE OF THE APPLICATION. For example querying some\_queue.empty? is fine, but assigning a value to a slot is not. We are monitoring the application, so do not be intrusive.
2. Be defensive in your before and after methods. Be prepared to handle unset values. If you forget to, your before or after method will raise an error and no before or after value is included in the event.
3. Avoid calling methods that (potentially) take a long time to execute. This will delay the application for the user.

## Testing

Test your integration module thoroughly before submitting it.

1. Check the dashboard Diagnostics Finetuning for your session ID to see whether you do not get discarded events of type rwa\_custom\_function. If you do, then your integration module generates more events then Diagnostics can process without impacting the system.
2. In addition your before and after methods do need time to run to gather the information you want. Make sure that these methods are as fast as possible, because this will be inserted in normal production operation. Any delay you incur here will translate directly into a slower application for the user.