ptc

integrity™ lifecycle manager

Integrations Builder Guide

11.1

Contents

1

Introduction

The *Integrity Lifecycle Manager Integrations Builder Guide* supplies information about building integrations using the Integrity Lifecycle Manager application programming interface (API). It provides an overview of the Integrity Lifecycle Manager API, instructions and guidelines to help you start coding your integration, and details about setting up Integrity Lifecycle Manager for the API.

Specifically, this chapter discusses the following topics:

- About this Guide on page 8
- Assumptions on page 8

About This Guide

This guide includes the following chapters:

Chapter 2: Understanding the Integrity Lifecycle Manager API on page 9

Provides information to help you understand how the Integrity Lifecycle Manager API can meet your integration needs. Before proceeding with developing your integration, it is useful to read this chapter to understand some of the concepts and features of the Integrity Lifecycle Manager API.

Chapter 3: Using the Integrity Lifecycle Manager API on page 19

Provides information to help you start developing your integrations using the Java and C versions of the Integrity Lifecycle Manager API

Chapter 4: API Conversion Compatibility on page 77

Provides information on understanding Integrity Lifecycle Manager API version compatibility.

Chapter 5: Published API Commands on page 97

Provides work item and result information for all published API commands.

Chapter 6: Administering the Integrity Lifecycle Manager API on page 191

Provides information on how to set up and administrate the Integrity Lifecycle Manager API.

Assumptions

Before using the Integrity Lifecycle Manager API, PTC assumes the following about your knowledge and experience:

- You understand Java, if you are building an implementation using the Java version of the Integrity Lifecycle Manager API.
- You understand C, if you are building an implementation using the C version of the Integrity Lifecycle Manager API.
- You understand the applications you are integrating with Integrity Lifecycle Manager.
- You understand the hardware platforms and operating systems that your applications are installed on and that Integrity Lifecycle Manager is installed on.
- You are comfortable with the command line interface (CLI) for Integrity Lifecycle Manager.

2

Understanding the Integrity Lifecycle Manager API

PTC recognizes the value of extending the functionality of Integrity Lifecycle Manager by providing a programmatic means of integrating with other tools and applications. The value of the Integrity Lifecycle Manager product is in its ability to interact with your source and processes at their point of origin. This often involves integrating with a third-party tool, or an internally developed tool or solution, and can span multiple platforms and implementation languages.

To develop integrations and solutions for your specific tool or process, you need the appropriate mechanism. Integrity Lifecycle Manager provides multiple integration mechanisms:

Integrity Lifecycle Manager Application Programming Interface (API)

This integration mechanism provides Java and C access for running Integrity Lifecycle Manager commands and working with their responses. Complete details are in this manual.

Command Line Interface (CLI) and Scripts

A complete set of commands is available for Integrity Lifecycle Manager. These allow a user, script, or program with access to a command processor to interact with Integrity Lifecycle Manager. Scripts are often easy to write for administrators and familiar to users. However, if a script requires parsing the command output, consider using the Integrity Lifecycle Manager API, which requires no parsing.

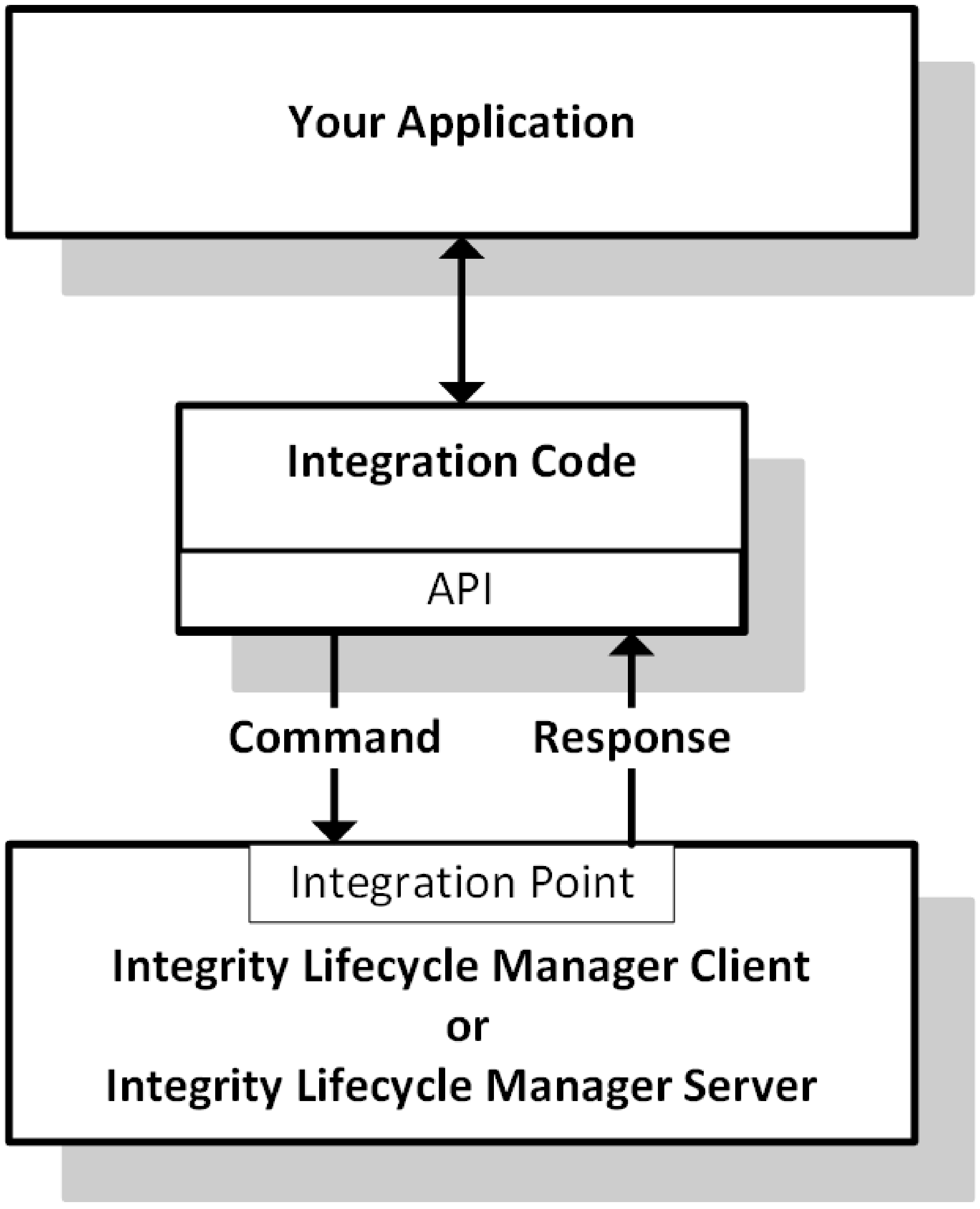
Event Triggers

Integrity Lifecycle Manager supports event triggers. Actions within either product can trigger any function that has been defined as a trigger. See the *Integrity Lifecycle Manager Installation and Upgrading Guide* for details.

This guide focuses on the Integrity Lifecycle Manager Java and C API, and its usage. This chapter contains the following information to help you understand how the Integrity Lifecycle Manager API can meet your integration needs.

The API Model

The Integrity Lifecycle Manager API uses a generic model to minimize the development and administration of all integrations built using the API. All integrations interact with Integrity Lifecycle Manager products through a generic command processor called the command runner. To run any command, a command is specified using its CLI syntax. A command runner sends the command to an Integrity Lifecycle Manager client or Integrity Lifecycle Manager, server which executes the command and returns a response. The integration can then access the response contents.



The same command runner is used to run any command for any application, only the command and its options change. All commands use the same response type. Rather than learning multiple APIs, there is only a single API to become familiar with.

The Integrity Lifecycle Manager API consists of three components:

Commands

The Integrity Lifecycle Manager CLIs define the functions that are available through the API. Key Integrity Lifecycle Manager commands (item and change package creation and manipulation) and core functions, used by most IDE (Integrated Development Environment) integrations, are published for API support. Limited default behavior is available for all commands. For more information, see Published API Commands on page 97.

Integration Points

The integration point is the point in Integrity Lifecycle Manager where integrations can connect and run commands. You connect and interact with Integrity Lifecycle Manager by running commands on the Integrity Lifecycle Manager client or the Integrity Lifecycle Manager server. The same commands are used for both integration points, although the behavior and valid options can change. Through this integration point, an integration establishes an API session with a specified API version. Commands are then executed within this session. For more information about sessions, see Sessions on page 30.

Access Methods

The access method is the technology you use to run the API. The access method does not contain any application logic. Currently, you can use Java or C as access methods. For more information on using the Java or C API, see Using the Integrity Lifecycle Manager API on page 21.

Answering the following questions helps to characterize a specific integration:

- How does the integration use the API model?
- What commands do the integration run?
- What are the integration points?
- What access method does the integration use?

For example, a typical IDE integration runs Integrity Lifecycle Manager commands on the Integrity Lifecycle Manager client (integration point) using Java (access method). A help desk integration could run Integrity Lifecycle Manager commands on the Integrity Lifecycle Manager server (integration point) using C (access method).

Integration Scenarios

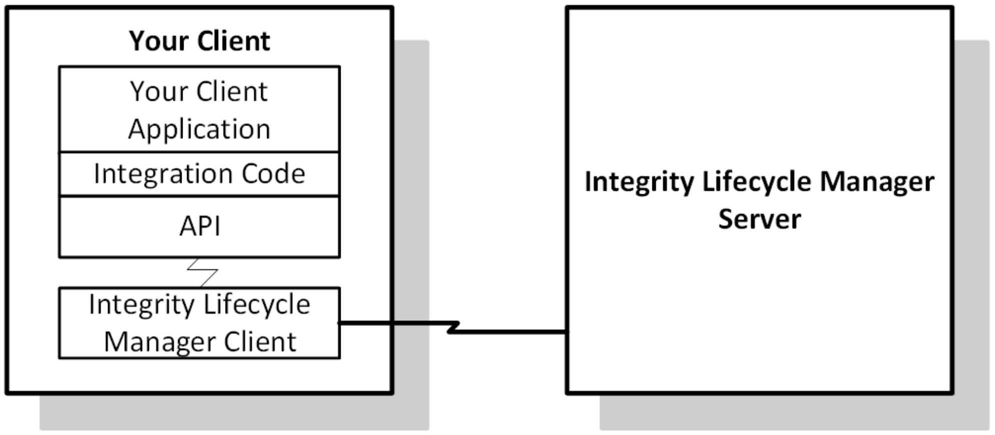
It is relatively easy to determine the commands that you want to run and the access method that you want to use for your integration. However, deciding which integration point to use is much harder. The Integrity Lifecycle Manager client and Integrity Lifecycle Manager server integration point behaves differently, depending on the location of the integration relative to the integration point.

The most common scenarios are described in these sections:

- Integrating with the Integrity Lifecycle Manager Client on page 13
- Integrating with the Integrity Lifecycle Manager Server on page 14
- Integrating with the Integrity Lifecycle Manager Client Server on page 16
- Integrating Your Desktop with the Integrity Lifecycle Manager Server on page 17

Integrating with the Integrity Lifecycle Manager Client

An Integrity Lifecycle Manager client integration is the most common integration. For example, it is used by all IDE and developer productivity tool integrations.


Your Client
Your Client Application
Integration Code
API
Integrity Lifecycle Manager Client
Integrity Lifecycle Manager Server

The Integrity Lifecycle Manager client integration has many unique features.

GUI Interaction for Status, Dialogs, and Views

Integrations can use the Integrity Lifecycle Manager client GUI when integrating with an Integrity Lifecycle Manager client. For more information, see Using the Graphical User Interface on page 54.

Common Session

By using a common session, an Integrity Lifecycle Manager client integration can provide the user with seamless interaction between the integration, Integrity Lifecycle Manager client GUI, and the CLI. Changes made through the GUI or

CLI can influence the integration and conversely. For example, establishing a connection using the Integrity Lifecycle Manager client provides the default connection for the integration. You also have the option of not using the tight coupling provided by a common session, and using the GUI and CLI independently from the integration.

User Preferences

All user preferences defined on an Integrity Lifecycle Manager client are available to an Integrity Lifecycle Manager client integration. This is an especially valuable feature. It allows an Integrity Lifecycle Manager client integration to not have to specify connection or user information. Instead, the integration relies on the defaults set on the Integrity Lifecycle Manager client.

Detecting the Integrity Lifecycle Manager Client Port

The port of the Integrity Lifecycle Manager client integration point can be automatically detected when integrating to an Integrity Lifecycle Manager client installed on the same system as the integration.

Local File System Access

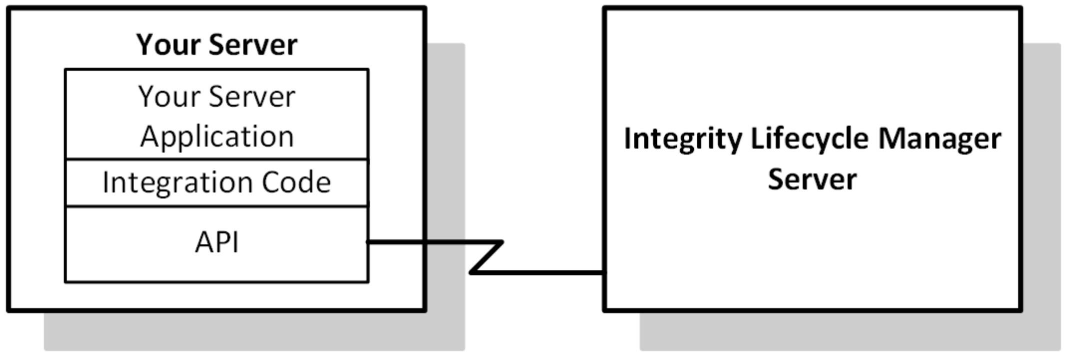
Commands run through the API on an Integrity Lifecycle Manager client have complete file system access, the same as if they were run directly on the Integrity Lifecycle Manager client. For more information, see Using File References on page 57.

Automatically Starting the Integrity Lifecycle Manager Client

The Integrity Lifecycle Manager client can be automatically started if it is not already running when integrating to an Integrity Lifecycle Manager client installed on the same system as the integration.

Integrating with the Integrity Lifecycle Manager Server

When no unique Integrity Lifecycle Manager client functions are required, you can integrate directly with the Integrity Lifecycle Manager server. Integrating with the Integrity Lifecycle Manager server eliminates the need for an Integrity Lifecycle Manager client installation and the administration that is associated with it. This scenario works well when integrating two server-based applications. For example, it works well when integrating a help desk with Integrity Lifecycle Manager.



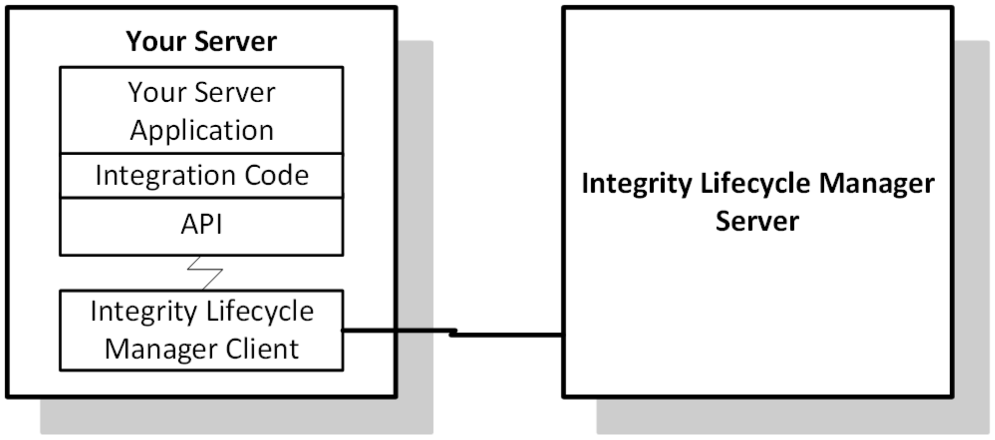
All commands are available. However, there are some restrictions on Integrity Lifecycle Manager client-oriented commands and options. For example, directing any output to a GUI causes an error. In addition, commands that contain a file reference satisfy that reference using the Integrity Lifecycle Manager server file system. For example, checking out a file checks it out on the Integrity Lifecycle Manager server. For more information, see Using File References on page 57.

Note

Integration with an Integrity Lifecycle Manager server that is set up as a proxy server is not supported.

Integrating with the Integrity Lifecycle Manager Client Server

Some server to server integrations require functions that are only available with the Integrity Lifecycle Manager client. If this is the case, you can install the Integrity Lifecycle Manager client on your application server. The Integrity Lifecycle Manager client can run simultaneous commands for multiple users.


Your Server
Your Server Application
Integration Code
API
Integrity Lifecycle Manager Client
Integrity Lifecycle Manager Server

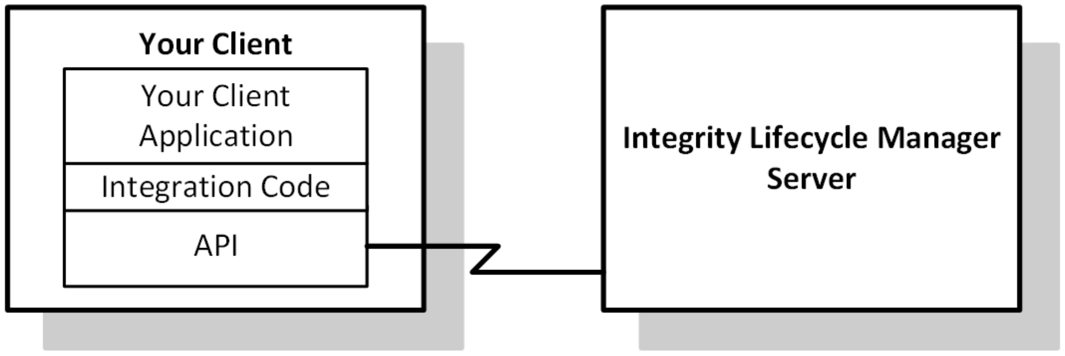
There are two main reasons for using this scenario:

- You need full local file system access, which is typically for Integrity Lifecycle Manager integrations.
- There is an intervening Integrity Lifecycle Manager proxy between the servers.

Change your connection and authentication policies to accommodate this scenario. For more information, see Setting Up Policies on page 192.

Integrating Your Desktop with the Integrity Lifecycle Manager Server

If your desktop application only needs information from the Integrity Lifecycle Manager server and nothing unique to the Integrity Lifecycle Manager client, then you can integrate your desktop directly to the Integrity Lifecycle Manager server.



This scenario requires the minimum configuration and administration.

Where to Go Next

Follow the steps in the table to develop an integration using the Integrity Lifecycle Manager API.

To Learn About This	See This
Guidelines and best practices for developing your integration	Using the Integrity Lifecycle Manager API on page 19
Supporting your integration across multiple versions of Integrity Lifecycle Manager	API Version Compatibility on page 77
Commands that you can use in your integration	Published API Commands on page 97
Setting up Integrity Lifecycle Manager for your integration	Administrating the Integrity Lifecycle Manager API on page 191
Information about public classes for Java API	Integrity Lifecycle Manager Java API documentation accessed through the Integrity Lifecycle Manager server Web page
Information about public header files for C API	Integrity Lifecycle Manager ANSI C API documentation accessed through the Integrity Lifecycle Manager server Web page

To Learn About This	**See This**
Response and behavior of commands when run through the Java API	`com.mks.api.util.APIViewer` in the Integrity Lifecycle Manager Java API documentation accessed through the Integrity Lifecycle Manager server Web page
Response and behavior of commands when run through the C API	`mksAPIViewer`, found in the *`Integrity client install directory`*`\bin` directory

3

Using the Integrity Lifecycle Manager API

This chapter contains information to help you start developing your integrations using the Integrity Lifecycle Manager API. Specific procedures and examples for the Java and C versions of the API are provided where appropriate.

The Java API is contained within `mksapi.jar`, which is installed in the lib directory on both the client and server.

- On the client, this JAR file is installed in`<install directory>\lib`.
- On the server, this JAR file is installed in `<install directory>\server\mks\lib`.

Make sure that `mksapi.jar` is available on your classpath.

The C API header files are installed on the Integrity Lifecycle Manager client in the `<install directory>\lib\include` directory.

This chapter contains the following information:

API Documentation

Details of all Java public classes and all C functions are provided in separate documents.

Java API Documentation

The Java API public classes are documented in the Integrity Lifecycle Manager Java API documentation, which can be accessed through the Integrity Lifecycle Manager server Web page. If this documentation is not available through the Web page, contact your administrator.

The Integrity Lifecycle Manager Java API documentation contains the following packages:

- `com.mks.api` contains classes and methods used to run API commands
- `com.mks.api` contains classes and methods used to access API responses.
- `com.mks.api.si` contains model type constants for Integrity Lifecycle Manager configuration management functionality.
- `com.mks.api.im` contains model type constants for Integrity Lifecycle Manager workflows and documents functionality.
- `com.mks.api.ic` contains model type constants for the Integrity Lifecycle Manager.
- `com.mks.api.util` contains classes and methods used for testing API commands.
- `com.mks.api.ext` contains classes and methods used to create integration command extensions.

C API Documentation

The C API functions are documented in the Integrity Lifecycle Manager ANSI C API documentation, which can be accessed through the Integrity Lifecycle Manager server Web page. If this documentation is not available through the Web page, contact your administrator.

The Integrity Lifecycle Manager ANSI C API documentation contains the following packages:

- `mksCommand.h` contains functions used to run API commands.
- `mksError.h` contains the error codes that API commands can return.
- `mksLog.h` contains functions used to log API messages.
- `mksResponse.h` contains functions used to access API responses.
- `mksResponseUtil.h` contains functions used for displaying API responses.

- `mksVersion.h` contains the Integrity Lifecycle Manager API version.
- `mksapi.h` includes all of the above.

Setting Up Your Environment

The following environment setup is required for the Integrity Lifecycle Manager API.

Java API Environment

You can use the Java version of the API with any platform that has a supported version of the Java Runtime Environment (JRE). For details on the supported versions, see API Versions on page 24.

Note

PTC recommends using JRE 1.6.

To use the Java API libraries on any UNIX platform, you must set the *LD_LIBRARY_PATH* environment variable as follows:

```
export LD_LIBRARY_PATH=installdir/lib/os
```

Where:

- *installdir* is the path to the Integrity Lifecycle Manager client install location
- *os* is either `solaris` or `linux`, according to the operating system on the Integrity Lifecycle Manager client machine

Note

The JRE installed with the Integrity Lifecycle Manager client and Integrity Lifecycle Manager server is intended for those applications only. PTC reserves the right to change the JRE version included in the client and server installations. To avoid incompatibility issues, include a JRE/JDK installation in each of your integrations. This ensures that your integrations are not impacted if the client or server is upgraded.

C API Environment

The C version of the API is implemented using `ANSI C`. You can use the C API with any `ANSI C` compiler on supported platforms. When compiling on a Windows platform, you must use the `_stdcall` calling convention. For details on the supported platforms, see the *Integrity Lifecycle Manager Help Center*.

Note

You cannot use the C API with Integrity Lifecycle Manager products before version 2005.

To use the C API libraries on any UNIX platform, you must set the environment variable *`LD_LIBRARY_PATH=$CLIENT_DIR`*`/lib/$sys`.

For North American UNIX servers to view German characters through the C API, the locale must be set correctly using `setlocale()`. If the locale is not set correctly, German characters come out as question marks (?). To correctly set the locale, do one of the following:

- In a shell, `#export LC_ALL=en_US.ISO8859-1`
- In a C program, `setlocale(LC_ALL, "");`

The C API uses the `OpenSSL` package to communicate with the Integrity Lifecycle Manager server via SSL when using a secure integration point. This package requires the `/dev/[u]random` device. If your platform does not have this device, one of the following packages can be installed and used:

- EGD: The Entropy Gathering Daemon (http://egd.sourceforge.net)
- PRNGD: Pseudo Random Number Generator Daemon (http://prngd.sourceforge.net/)

Integration Objects

Before starting to develop your integration, you must understand the following objects:

- API Versions on page 24
- Integration Points on page 26
- Sessions on page 30
- Command Runners on page 32

API Versions

When an integration communicates with Integrity Lifecycle Manager, it requests and specifies the version that the Integrity Lifecycle Manager API is to use. The API version provides the integration with independence and a stable API as newer versions of Integrity Lifecycle Manager become available. If Integrity Lifecycle Manager supports the API version requested by the integration, Integrity Lifecycle Manager provides the necessary API compatibility support to ensure that the integration's use of the API remains stable.

Supported API Versions

The following table lists the available versions of the Integrity Lifecycle Manager API:

API Version	Description
4.16	As part of localization support, exposes a display name for some admin objects using the Java API method `Item.getDisplayId().`
4.15	Supports `APIWarningException` being reported in appropriate views, so that the Java API does not throw that exception when obtaining the API response.
4.14	The `-showhistory` option of the `im viewissue` command now displays Created By rather than Modified By for the first entry in the history.
4.13	Support for distinguishing literal null values from null values returned by ambiguous computed field values in workflows and documents items. An API item of `im Computation` model type now represents computed field values. The type now contains the value `isAmbiguous` or `isDynamic`. **Note** With this update, older versions of the Integrity Lifecycle Manager API can return null values with no context for simple data types. Support for including versioned items, as specified by an Integrity 10.5 query definition. **Note** Integrity API 4.12 or older excludes versioned items by default. The client/command removes versioned items from the query results. There is no change to operation of the query on the server and no modification to the query definition sent to the server.

API Version	Description
4.12	Support for Integrity Lifecycle Manager field and value display strings from the Java API. Support for Integrity Lifecycle Manager image data from the Java API.
4.11	Improved API details on Integrity Lifecycle Manager multi-valued user fields (changed from a list of strings to a list of API items representing users).

4.10	API support for PTC Gateway.
	Improved API details for Integrity Lifecycle Manager FVA fields.
	Deprecated the following function calls:
	`createIntegrationPoint()`, `createIntegrationPoint (String host, int port)`
	and
	`createIntegrationPoint (String host, int port, boolean secure)`
	and replaced with the following function calls:
	`createIntegrationPoint (int apiMajorVersion, int apiMinorVersion)`, `createIntegrationPoint (String host, int port, int apiMajorVersion, int apiMinorVersion)`
	and
	`createIntegrationPoint (String host, int port, boolean secure, int apiMajorVersion, int apiMinorVersion)`

4.9	Initial published version of the Integrity Lifecycle Manager API.

Supported Version Mapping

The following table lists the supported versions of the Integrity API:

Integrity Version	API Version Support
10.8 and above	Integrity API 4.10 to 4.16
10.7	Integrity API 4.10 to 4.15
10.6	Integrity API 4.10 to 4.14
10.5	Integrity API 4.10 to 4.13
10.4	Integrity API 4.10 to 4.12
10.0 to 10.3	Integrity API 4.9 to 4.11

Integrity Version	API Version Support
2009	Integrity API 4.9 and 4.10
2007	Integrity API 4.9

Tip

Each version of Integrity Lifecycle Manager indicates the highest API version that it supports. For more information, see API Version Compatibility on page 77.

Installing and Bundling the API

The Integrity Lifecycle Manager API essentially becomes part of the integration you develop; therefore, PTC recommends that you copy and install/bundle the API with the integration. It is not a good practice to simply reference the API support `JARs/DLLs` located in the Integrity Lifecycle Manager installation directory, as these are subject to change.

Note

You want to copy and bundle the `mksapi.jar` file with your integration. If you point to the `mksapi.jar` file in the Integrity Lifecycle Manager client or Integrity Lifecycle Manager server installation instead, you potentially introduce the following risks:

- Binary incompatibilities only detected at runtime
- Behavioral incompatibilities

If you point to the `mksapi.jar` file, you must test your integration each time the `mksapi.jar` file changes. This can occur during installation of a new version through the installer, a service pack application, or a hotfix application.

Integration Points

The integration point represents the Integrity Lifecycle Manager client or Integrity Lifecycle Manager server you want to connect to. The API makes requests of an integration point, and the integration point determines the response to the requests.

Controlling Access to an Integration Point

You control access to an integration point through connection and authentication policies.

The connection policy determines where an integration point accepts a connection from. By default, the client only accepts connections from `localhost` and the server only accepts connections from a specified set of IP addresses.

The authentication policy determines whether you must specify a user and a password for a session. By default, the client integration point allows connections without specifying a user (as long as connections are limited to `localhost`). The server default requires a user ID and password.

Note

- Certain special characters in user names sometimes cause the API connection to fail.
- Do not confuse the user and password required for a session with the user information specified on the command runner or through the `--user password` command options.

For more information on connection and authentication policies, see Setting Up Policies on page 192.

Communicating with an Integration Point

The API and the integration point communicate using TCP connections. The server uses the standard configured Integrity Lifecycle Manager port to listen for communications requests (7001 by default). The client also uses a configured port, which the API can automatically detect.

If you are integrating with the server, you can use a secure port for communications with the API. For information on creating an integration point that uses a secure port, see Creating an Integration Point on page 27. For information on how to configure a secure port, see the *Integrity Lifecycle Manager Installation and Upgrading Guide*.

Before you can use an integration point, you must create a session. Once you create a session, you receive any exceptions related to the integration point, and its connection and authentication policies.

Creating an Integration Point

An integration point must be created for each physically different client or server you want to integrate with.

To maintain compatibility between multiple versions of Integrity Lifecycle Manager, API compatibility is implemented through the communication protocol between the API and the Integrity Lifecycle Manager integration point. Through the Integrity Lifecycle Manager API, integrations can request command execution and command response, based on a fixed request version. The Integrity Lifecycle Manager integration point is contractually obligated to respect the given request.

PTC recommends that integrations explicitly specify the (API) request version expected and validated. For example:

```
IntegrationPointFactory factory = IntegrationPointFactory.getInstance();
 IntegrationPoint ip = factory.createLocalIntegrationPoint(4,10); …
```

In this Java-API code, the integration specifies that it expects the API's integration point to use request version 4.10. In this case, this is the contractual request version offered by Integrity Lifecycle Manager 2009). This is transmitted to Integrity Lifecycle Manager on each command call with the expectation that, by contract, Integrity Lifecycle Manager respects the request version. This means that it becomes the responsibility of newer versions of Integrity Lifecycle Manager to meet the original contract as indicated by request version 4.10.

Creating an Integration Point for the Java API

1. Create an instance of the `ItegrationPointFactory` class by calling `IntegrationPointFactory.getInstance().`
2. Create an integration point using one of the following methods, depending on whether you are connecting to a local client or a remote client or server:
   - Assume that you are connecting to a local client. Create an `IntegrationPoint` by calling the `createlocalIntegrationPoint(int apiMajorVersion, int apiMinorVersion)` method from the `IntegrationPointFactory` instance. The `int apiMajorVersion` and `int apiMinorVersion` parameters specify the major and minor number of the requested API version.

     You can also have the API start the Integrity Lifecycle Manager client f it is not already running. For more information, see the Integrity Lifecycle Manager Java API documentation.
   - Assume that you are connecting to a remote server or client. Ccreate an `IntegrationPoint` by calling the `createIntegrationPoint(String host, int port, int apiMajorVersion, int apiMinorVersion)` method from the `IntegrationPointFactory` instance. The `String host` parameter is the host name. The `int port` parameter is the port number of the Integrity Lifecycle Manager client or Integrity Lifecycle Manager server to which to connect. The `int apiMajorVersion` and `int apiMinorVersion` parameters specify the major and minor number of the requested API version.

Creating an Integration point for the C API

1. Call the `mksAPIInitialize ()` function.
   - Assume that you are connecting to a local client. Create an `mksIntegrationPoint` by calling the `mksCreateLocalIntegrationPoint(mksIntegrationPoint *ip, unsigned short apiMajorVersion, unsigned short apiMinorVersion, unsigned short autostart)` function. The `unsigned short apiMajorVersion` and `unsigned short apiMinorVersion` parameters specify the major and minor number of the requested API version.

Note

You can only initialize and terminate the C API once.

2. Create an integration point using one of the following functions, depending on whether you are connecting to a local client or a remote client or server:
   - Assume that you are connecting to a remote server or client. Create an `IntegrationPoint` by calling the `mksCreateIntegrationPoint(mksIntegrationPoint *ip, char *host, unsigned int port, unsigned short secure, unsigned short apiMajorVersion, unsigned short apiMinorVersion, unsigned short autostart)`. The `host` parameter is the host name. The `unsigned int port` parameter is the port number of the Integrity Lifecycle Manager client or Integrity Lifecycle Manager server to which to connect. The `unsigned short apiMajorVersion` and `unsigned short apiMinorVersion` parameters specify the major and minor number of the requested API version.

Note

If you want to create an integration point using a secure communications protocol, call the `mksCreateIntegrationPoint` function with the secure parameter set to true. For information on setting up your environment for a secure integration point, see C API Environment on page 23

Sessions

Commands are run within a session on the integration point. A session is established in an integration point at a specified API version. Sessions control the interactions of commands being executed on the same integration point. A session has a physical TCP connection between the API and the integration point. A session can run any command on any server for any application, for any user that the integration point allows.

The main factor in determining how to use sessions is the number of users that the integration is serving. For more information, see Managing Users on page 60.

Session Caching

A session caches all application information per user. For example, the Integrity Lifecycle Manager client caches recent sandbox information for quickly retrieving it again. A session also caches server connection and state information shared by all users of the session.

On the Integrity Lifecycle Manager, client a session caches all server connections between the Integrity Lifecycle Manager client and Integrity Lifecycle Manager server. Each connection is stored from the time that it is made until either the session is deleted or the individual connection is disconnected. When a command is run, an existing server connection is used if there is one for the specified server, port, and user. The password is not required nor is it checked.

A session maintains the current state of commands. For example, a session maintains the current values of command preferences. The current state of a preference can be changed through the GUI preferences or the `setprefs` command. Once a change occurs, the new preference is used for every subsequent command run by any user on this session.

Using a Common Session

A special common session is available only on the Integrity Lifecycle Manager client. The common session is the standard Integrity Lifecycle Manager session available to anyone using the Integrity Lifecycle Manager client. It is shared with the CLI and the GUI, and it is available for integrations as well. This provides a consistent experience for users who interact with the Integrity Lifecycle Manager client through all three methods. The common session is not restricted to local clients.

Creating a Session

You create a session from an integration point. An integration ID can optionally be given to this session. This ID can be used within the product, such as within triggers on the Integrity Lifecycle Manager server, to help identify the command operations executed from a given integration. For more information on integration IDs, see article CS229260 on the PTC Integrity eSupport portal.

Creating a Session for the Java API

Create a `session` from the `IntegrationPoint` instance by doing one of the following:

- For a standard session on an Integrity Lifecycle Manager server, call the `createNamedSession()` method, passing the user and password. Optionally, you can pass an ID identifying the calling integration. You can also pass an API version if it is necessary to override the API version associated with the integration point.
- For a standard session on an Integrity Lifecycle Manager client, call the `createNamedSession()` method. Optionally, you can pass an ID identifying the calling integration. You can also pass an API version if it is necessary to override the API version associated with the integration point.
- For a common session on an Integrity Lifecycle Manager client, call the `getCommonSession()` function. No integration ID is associated with this session because it is commonly available to all integrations as well as the Integrity Lifecycle Manager clients own CLI and GUI operations.

Creating a Session for the C API

Create a `session` from the `mksIntegrationPoint` variable by doing one of the following:

- For a standard session on an Integrity Lifecycle Manager server, call the `mksCreateSession()` function, passing the user and password.
- For a standard session on an Integrity Lifecycle Manager client, pass `NULL` as user and password arguments to the `mksCreateSession()` function.
- For a common session on an Integrity Lifecycle Manager client, call the `mksGetCommonSession()` function. No integration ID is associated with this session because it is commonly available to all integrations as well as the Integrity Lifecycle Manager clients own CLI and GUI operations.

Command Runners

The command runner manages the sequential execution of commands on a session. If more than one command must be executed at the same time on an integration point, you have to create multiple command runners for a session. With multiple sessions, it is possible to have an integration using multiple command runners, concurrently running commands in multiple sessions, connected to multiple integration points.

You can set connection-based command options for a command runner that are applied to every command. For more information, see Managing Users on page 60.

Note

Even though not all application commands explicitly allow these connection settings, they do not cause an error if set for any command.

Creating a Command Runner

You create a command runner from a session.

- For the Java API

  Create an instance of the `CmdRunner` class by calling the `createCmdRunner()` method from the `Session` instance.

- For the C API

  Create an instance of the `mksCmdRunner` by calling the `mksCreateCmdRunner()` function from the *mksSession* variable

Identifying the Command Runner

By default a command runner is used to invoke commands in the context of the integration ID taken from its associated session. It is possible to override this context and supply a different ID to the command runner. This allows for the command runner to have its own invocation ID. This ID can then be used within the product to identify the context under which a command and its operations are being executed. This is useful when the common session is used.

- For the Java API

  Call the `setInvocationID()` method from the `CmdRunner` instance, passing an ID that can be used to identify the calling context for command execution.

- For the C API

Call the `mksCmdRunnerSetCallerID()` function, passing the command runner and an identifier string as arguments.

For more information on integration IDs, see article CS229260 on the PTC Integrity eSupport portal.

Running Commands

There are three activities related to running commands using a command runner:

- Specifying a Command on page 33
- Executing a Command on page 35
- Reading Responses on page 35

Specifying Commands

A command specifies the application to interact with and the command name, and can have any number of options and selections.

Anything that can be specified on a command line can be specified using a `Command` class or function.

For details on the options and selections for each command, see the CLI man pages. For details on the responses returned by each command, see Published API Commands on page 97.

Specifying a Command for the Java API

You can either specify the command with the `Command` class and related classes or through a `String` array.

For example, for the `si about` command, you can either pass the `String` array:

```
String[] cmd = new String[] {"si", "about"};
```

Or you can construct the command using the following:

```
Command cmd = new Command(Command.SI, "about");
```

For example, you can construct the `im issues` command with the following options:

```
Command issuesView = new Command(Command.IM, "issues");issuesView.addOption
( new Option("fields","ID,Type,Summary"));issuesView.addOption
( new Option("query", "Quick Query"));
```

Note

Unlike CLI commands, options for commands used in the API do not require a "–" prefix if the commands are created using the `Command` class. The prefix is still required if `String` arrays are used.

Specifying a Command for the C API

You can either specify the command by creating a char * array or through an `mksCommand` structure

For example, for the `im viewissue` command, you could either create a char * array:

```
wchar_t* cmd[] = {"im", "viewissue", "--gui"};
```

Or you can construct the command using an `mksCommand` structure:

```
mksCommand = cmd;
cmd = mksCreateCommand();
cmd->appName = "im";
cmd->cmdName = "viewissue";
mksOptionListAdd(cmd->optionList, "--gui", NULL);
```

Default Command Behavior

If a specific behavior is required of a command, PTC recommends specifying all the options needed in the command. This is a much better practice than relying on defaults set through command preferences.

You can use the GUI to specify command settings for any command run through the API by specifying the option `--settingsUI=gui`. For more information, see Using the Graphical User Interface on page 54.

Some preferences can only be changed through the CLI `si setprefs` command, using the `--ui` option. For more information, see `si viewprefs` and `si setprefs` in the CLI man pages.

There are also API-specific default preferences that are used in special situations. For example, the default set of fields shown in a view command in the API are specially defined as API defaults. These fields are not subject to changes that are possibly reflected in the GUI.

Executing a Command

The command runner executes the command and passes it to the integration point. The integration point runs the command and returns a response, which contains all information related to the execution of the command. The response contents can then be accessed or another command can be run. Each command execution generates a new response. For more information on accessing response contents, see Reading Responses on page 35.

If a command specified for the Java API ends in an error, a command level exception is thrown and should be caught. For more information, see Catching Exceptions on page 45.

Reading Responses

A response contains all of the output from executing a command. All commands communicate using the same response type, but they use the response differently, depending on the type of command and what it has to communicate. This section describes the response for the creation, action, and view command types.

Note

Often it is useful to know the host, port, and user that a command used to connect to the Integrity Lifecycle Manager server. The Integrity Lifecycle Manager API has functions and methods to return this information. For more information, see the Integrity Lifecycle Manager Java API documentation or the Integrity Lifecycle Manager ANSI C API documentation.

To find any errors that occurred as part of the response for a command specified for the C API, you must search for them explicitly. For more information, see C API Application Errors on page 50.

Creation Commands

Creation commands create something new, for example, an Integrity Lifecycle Manager item. The command names usually start with the word `create`. The response for a creation command communicates the item that was created or what errors prevented the item from being created.

If the command ran successfully, the result contains information about what was created. The result message is the same text shown by the CLI. For example, `Created issue 123`.

To continue working with the created item, you do not need this message; you only need the ID of what was created. The following examples show how to retrieve the item ID from the result.

Java Example: Retrieving the Item ID

```
Response response = cmdRunner.execute(myCreationCmd);
String newId = response.getResult().getPrimaryValue().getId(
```

C Example: Retrieving the Item ID

```
wchar_t newId[256];

response = mksCmdRunnerExecCmd(cr, cmd, NO_INTERIM);
result = mksResponseGetResult(response);
item = mksResultGetPrimaryValue(result);
rtn = mksItemGetId(item, newID, sizeof(newId));
```

Action Commands

Action commands perform operations within the application. For example, action commands exist for editing, deleting, checking out, checking in, and resynchronizing. The response for an action command communicates the following:

- Item on which the action was attempted
- Result of the requested action, which can be either the change that occurred or an error that occurred while trying to change the item

Unlike creation commands, which only create a single item, action commands can work with multiple items. The response for all action commands contains result and exception information for each item that was acted on. This allows for exact reporting of the item acted on, and the result or exception from that item. If an exception is present on any item, the command itself reports the failure with an exception of its own.

If the command runs successfully on a single item, you must know from the response that no errors occurred. Ensure that you know what the command reports before making this assumption. In other cases, retrieving more information from the response can be necessary.

Retrieving Work Items

Work items indicate the items that the command processed. For example, running `si resync` without specifying a selection resynchronizes all members in a sandbox and reports on each. The members can be retrieved from the command response by their work item ID, or through iteration as in the following examples.

Java Example: Retrieving Work Items

```
String newRevisionId =
workItem.getResult().getPrimaryValue().getId();
```

C Example: Retrieving Work Items

```
response = mksCmdRunnerExecCmd(cr, cmd, NO_INTERIM);
workItem = mksResponseGetFirstWorkItem(response);/
* Do something with the first work item. */
while ((workItem = mksResponseGetNextWorkItem(response)) != NULL)
{
/* Do something with the work item. */
}
```

Each retrieved work item contains an ID, a context (if needed), a display string, a model type (for example, state, issue), and, optionally, a result.

Retrieving Work Item Results

Each work item optionally contains information about the action taken. For example, assume that you run `si resync` and that there is nothing to resynchronize for a particular member. There is no result for that work item because no action was taken. If an action was taken, there is a result. In the case of `si resync`, the result is the revision that was checked out. The following examples show how to retrieve result information for a work item.

Java Example: Retrieving Work Items

```
String newRevisionId = workItem.getResult().getPrimaryValue().getId();
```

C Example: Retrieving Work Items

```
result = mksWorkItemGetResult(workItem);
item = mksResultGetPrimaryValue(result);
newId = (wchar_t *) malloc(256*sizeof(wchar_t));
rtn = mksItemGetId(item, newId, 256 * sizeof(wchar_t));
```

View Commands

View commands provide access to the current state of information within an application. Typically, the command names are either the name of the item type to be viewed (for example, issues, sandboxes), or they start with the word view. The response for a view command communicates the content of the item being viewed, no matter how complex.

Like action commands, view commands always return work items, command exceptions, and work item exceptions. Unlike action commands, a view command does not need to report the results of an action.

Retrieving View Information

Work items can be retrieved from the response for view commands in the same way as they are for action commands.

Once you have identified the work items, you can access their content. All work item content is contained in fields. You can access the information in the fields without parsing. The fields can be basic data types, such as string and Boolean. Or, they can contain a list of basic data type values, items, or a list of items.

For example, a work item is a type of item. These items can, in turn, contain fields, which can contain items or a list of items. In this way, the response for a view command becomes a hierarchy of information that can be accessed easily.

To access view information from the response, you must get the field from the item then read its value. If the field contains another item, you can get that item's ID, context, or model type, or any of its fields, and so on. You can use an iterator over the fields. The field value is null with a data type of null if the contents are not set within the application.

The following examples show how to retrieve a field value from a sample Integrity Lifecycle Manager item.

Java Examples: Retrieving View Information

Example 1: Retrieving the Created Date for the Item

```
Date myDate = workItem.getField("Created Date").getDateTime();
```

In addition to retrieving the field as the matching data type, you can use the `getValueAsString()` method to return any data type as a string. This is shown in the next example.

Example 2: Retrieving the Created Date for the Item as a String

```
String myDateString = workItem.getField( "Created Date").getValueAsString();
```

For the basic data types, using the `getValueAsString()` method just returns the string representation. For fields that are items, it returns the item ID. See the Integrity Lifecycle Manager Java API documentation for details of the behavior of this method.

Example 3: Retrieving the Item State

```
String state = workItem.getField("State").getValueAsString();
```

Example 4: Retrieving the Assigned User for the Item

```
Item user = (Item)workItem.getField("Assigned User").getItem();
String fullname = user.getField("fullname").getValueAsString();
```

Example 5: Retrieving the List of Item Relationships

```
Field related = workItem.getField("MKSIssueRelationships");
Item first = (Item)related.getList().get(0);
String firstId = first.getId();
String firstState = first.getField("State").getValueAsString();
```

C Examples: Retrieving View Information

Example 1: Retrieving the Created Date for the Item

```
time_t myDate;
mksrtn rtn;
mksField field;
field = mksWorkItemGetField(workItem, L"Created Date");
rtn = mksFieldGetDateTimeValue(field, &myDate);
```

In addition to retrieving the field as the matching data type, you can use the `mksFieldGetValueAsString()` function to return any data type as a string. This is shown in the next example.

Example 2: Retrieving the Created Date for the Item as a String

```
wchar_t myDateString[256];
mksrtn rtn;

field = mksWorkItemGetField(workItem, L"Created Date");
rtn = mksFieldGetValueAsString(field, myDateString,
sizeof(myDateString));
```

For the basic data types, using the `mksFieldGetValueAsString()` method just returns the string representation; for fields that are items, it returns the item ID. See the Integrity Lifecycle Manager ANSI C documentation for details of the behavior of this function.

Example 3: Retrieving the Item State

```
wchar_t state[256];
mksrtn rtn;

field = mksWorkItemGetField(workItem, L"State");
rtn = mksFieldGetValueAsString(field, state, sizeof(state));
```

Example 4: Retrieving the Assigned User for the Item

```
wchar_t fullname[256];
mksItem user;
mksrtn rtn;

field = mksWorkItemGetField(workItem, L"Assigned User");
rtn = mksFieldGetItemValue(field, &user);

field = mksItemGetField(user, L"fullname");
rtn = mksFieldGetValueAsString(field, fullname, sizeof(fullname));
```

Example 5: Retrieving the List of Item Relationships

```
mksItem first;
mksItemList itemList;
mksrtn rtn;

field = mksWorkItemGetField(workItem, L"MKSIssueRelationships");
rtn = mksFieldGetItemListValue(field, &itemList);

first = mksItemListGetFirst(itemList);
rtn = mksItemGetId(first, firstId, sizeof(firstId));

field = mksItemGetField(first, L"State");
rtn = mksFieldGetValueAsString(field, firstState,sizeof(firstState));
```

For fields that contain a list of values or a list of items, you can extract a known element, which is shown in this example. You can also iterate over the full list of items.

Command Response Summary

The following table shows the generic response information for typical command executions.

Command Type	Response Pattern	Notes
Creation command—successful	• Application and command • Exit code of 0 • Result with created item ID and optional context	Sample commands: • `im createissue` • `im createsandbox`
Action command—successful	• Application and command • Exit code of 0 • Work items with type and ID • Results with affected item ID and optional context	Sample commands: • `si co` • `si ci`
View command—successful	• Application and command • Exit code of 0 • Work items with type and ID ○ `Fields` ○ `Item Lists` ○ *n* Items with ID	Sample commands: • `im viewissue` • `im users`

Command Type	Response Pattern	Notes
	◦ *n* Fields ◦ *n* Item Lists	
Command succeeded, but with warnings	• Application and command • Exit code of 23 • API exception	This exit code is used when a command has succeeded, but important information must be communicated for one or more work items. Sample command: • `si deactivatedevpath`
Command failed	• Application and command • Exit code of non-0 • API exception	This applies to all types of commands.
Work item error	• Application and command • API exception • Exit code of non-0 • Work items with type and ID ◦ `API Exception`	This applies to action and view type commands. The exception is on the work item that encountered the error. Work items that did not encounter an error have work item contents.

Viewing Responses with the API Viewer Utility

The response contents for each command depend on the command type and the specific command options used. The Integrity Lifecycle Manager API Viewer Utility allows you to run any command on an Integrity Lifecycle Manager client or Integrity Lifecycle Manager server. The complete contents of the response is returned to help you understand the API and how to read a particular response.

PTC strongly recommends that you use this utility to run commands that are used in an integration. This ensures that you are familiar with their responses and behaviors through the API.

Running the API Utility for Java

To run the API Viewer utility for the Java API, run the command:

```
java –jar mksapi.jar …
```

Where `. . .` specifies any desired options or arguments.

If you run without options or arguments, a CLI-stylized usage message displays the various options that can be used. Because the command line accepts CLI-stylized options, you no longer have to use Java system properties. However, if you prefer using them, consult the Integrity Lifecycle Manager Java API documentation for `com.mks.api.util.APIViewer`.

Assume that you run the following command through the Integrity Lifecycle Manager API Viewer:

```
im issues "--fields=Summary,Type,Created Date" 13 14
```

The following output could be produced:

```
Response:
App. Name    = im
Command Name = issues
Work Item:
Id          = 13
Context     = NULL
Model Type = im.Issue
Field:
Name      = Summary
Data Type = wchar_t *
Value     = Table too large, must be
            reworked.
Field:
Name      = Type
Data Type = mksItem
Item:
Id          = Bug
Context     = NULL
Model Type = im.Type
Field:
Name      = Created Date
Data Type = time_t
Value     = Fri Dec 05 10:33:24 CST 2003
Work Item
Id          = 14
Context     = NULL
Model Type = im.Issue
Field:
Name      = Summary
Data Type = wchar_t *
Value     = The panel isn't leaving enough
            room for the buttons.
Field:
Name      = Type
Data Type = mksItem
Item:
```

```
Id         = Bug
Context    = NULL
Model Type = im.Type
Field:
Name      = Created Date
Data Type = time_t
Value     = Wed Feb 04 10:33:43 CST 2004
Exit Code = 0
```

Running the API Utility for the C API

To run the API Viewer utility for the C API, run the `mksAPIViewer` application located in the `<Integrity client install directory>\bin\` directory. To view the usage commands, type `mksAPIViewer` in a command prompt or session window. Some notable options include:

- `--showconnection` returns the host, port, and user of the command used to connect to the Integrity Lifecycle Manager server.
- `--xml` displays all command output in XML format.

Assume that you run this command through the Integrity Lifecycle Manager API Viewer:

```
im issues "--fields=Summary,Type,Created Date" 13 14
```

It could produce the following output:

```
Response:
App. Name    = im
Command Name = issues
Work Item:
Id         = 13
Context    = NULL
Model Type = im.Issue
Field:
Name      = Summary
Data Type = wchar_t *
Value     = Table too large, must be
            reworked.
Field:
Name      = Type
Data Type = mksItem
Item:
Id         = Bug
Context    = NULL
Model Type = im.Type
Field:
Name      = Created Date
Data Type = time_t
Value     = Fri Dec 05 10:33:24 CST 2003
```

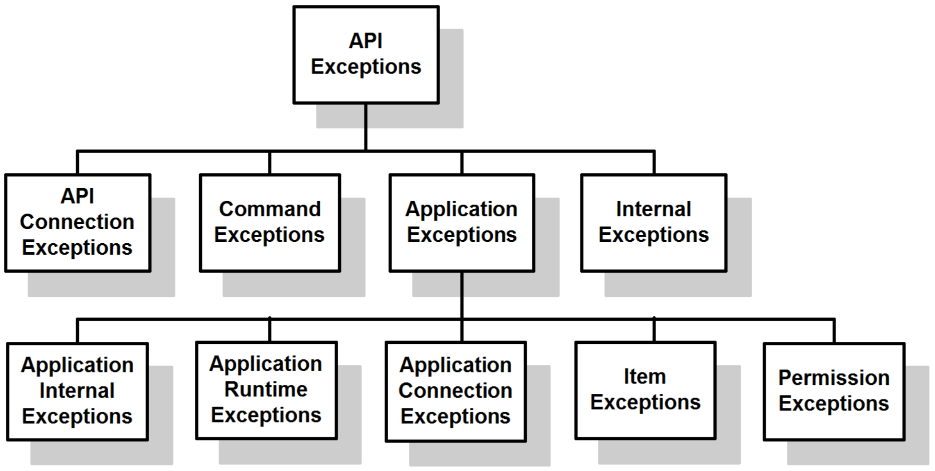
```
Work Item
Id          = 14
Context     = NULL
Model Type = im.Issue
Field:
Name       = Summary
Data Type = wchar_t *
Value      = The panel isn't leaving enough
             room for the buttons.
Field:
Name       = Type
Data Type = mksItem
Item:
Id          = Bug
Context     = NULL
Model Type = im.Type
Field:
Name       = Created Date
Data Type = time_t
Value      = Wed Feb 04 10:33:43 CST 2004
Exit Code = 0
```

Java API Exceptions

This section describes the exceptions that can be thrown while using the Integrity Lifecycle Manager Java API.

Integrity Lifecycle Manager Java API exceptions use a generic model, with classes for each general type of exception. The exception classes represent the following types of exceptions:

- All exceptions associated with an API connection
- All exceptions associated with a command definition
- All specific application exceptions and their attributes


API Exceptions
API Connection Exceptions
Command Exceptions
Application Exceptions
Internal Exceptions
Application Internal Exceptions
Application Runtime Exceptions
Application Connection Exceptions
Item Exceptions
Permission Exceptions

Catching Exceptions

If a command ends in error, a command level exception is thrown and must be caught. The exception class identifies the type of exception. The exception ID identifies the specific exception condition. This ID is accessed by using the `getExceptionId()` method on the `APIException`.

Specific exceptions can contain additional information that can further identify the nature of the error. You must read the response to get this information. You can access the final command exit code from the response.

Use the `getResponse()` method on the caught exception as shown in the following example:

```
Response response = null;
try {
response = cmdRunner.execute(myCmd);
// read response
}
catch (APIException ae) {
response = ae.getResponse();
}
```

Even when a command fails, it could have completed some work. For information on how to find out what work has been successfully completed, see Retrieving Work Items on page 36.

Use the API Viewer utility to see the information returned for specific error conditions. For more information on using the API Viewer, see Viewing Responses with the API Viewer Utility on page 41.

If you are using interim responses, commands level exceptions are not thrown when executing a command. For more information, see Using Interim Responses on page 55.

Work Item Exceptions

Work item exceptions are thrown when retrieving a work item from a response using the `WorkItemIterator`. When a work item exception is thrown, all command output is still available in the response. You can determine the work item you were attempting to retrieve when the exception was thrown through the `WorkItemIterator.getLast()` method. If you are not using the `WorkItemIterator`, use the `getAPIException()` method on a work item to see if there is an error associated with the item.

The exception class identifies the general type of exception. To get a more specific exception, you must use the `getExceptionId()` method to retrieve the exception ID. Specific exceptions can contain additional information on the nature of the error.

If the error occurs as a result of anything other than executing a command on the Integrity Lifecycle Manager client or Integrity Lifecycle Manager server, there is no exception ID.

API Connection Exceptions

API connection exceptions occur when a connection fails between the API and the integration point. The following table gives some examples of connection exceptions and the reason why they occur.

Exception Class	Reason for Exception
`InvalidHostException`	API cannot connect to configured integration point. There is no server/ client running at the configured address for integration point.
`InvalidIntegrationPointException`	API can connect to an address but the integration point details are invalid, for example, invalid host name or invalid port.
`IncompatibleVersionException`	Integration point does not support version of API that is attempting to communicate with it.

Command Exceptions

Command exceptions occur when a command does not execute.

Note

Complex command specifications can fail due to Integrity Lifecycle Manager validation errors. However, such errors can still be reported as command exceptions. Complex commands specifications are most frequently associated with filters, queries, and rules.

The following table gives some examples of command exceptions and the reason why they occur.

Exception Class	Reason for Exception
`CommandAlreadyRunningException`	Command runner already in use and cannot be used to run another command without first interrupting it.
`CommandCancelledException`	Command canceled due to a failed pre-condition. For example, you clicked **Cancel** when asked for settings.
`InvalidCommandOptionException`	Invalid command option used or command option missing.
`InvalidCommandRunnerStateException`	Command runner cannot be used to execute commands. For example, the command runner has been released.
`InvalidCommandSelectionException`	Invalid command selection used or command selection missing.
`UnknownCommandException`	Command unknown.
`UnsupportedFunctionException`	Error occurred while executing a command. For example, unsupported command feature such as `-nobatch` are used.
`UnsupportedVersionException`	API version not compatible with integration point.

Application Exceptions

Application exceptions occur when there is a processing or validation error within Integrity Lifecycle Manager.

The application exception classes represent only the generic application exceptions. There are many more specific exceptions that are possible within each generic application exception. Each specific exception is mapped to an exported exception name used for API exception reporting. The specific exception that occurred can be identified using the `getExceptionId()` method.

The three main types of API application exceptions are:

- Application connection exceptions
- Item exceptions
- Permission exceptions

The following table lists some general API application exceptions:

Exception Class	Reason for Exception
`ApplicationInternalException`	Internal error occurred on integration point while executing command.
`ApplicationRuntimeException`	Error occurred while integration point running command because of unanticipated application problem.
`UnsupportedApplicationException`	Application passed in with command unknown/invalid.

Application Connection Exceptions

Application connection exceptions occur when there is an error while initiating a connection to the Integrity Lifecycle Manager server.

The following table lists the application connection exceptions and the reason why they occur.

Exception Class	Reason for Exception
`IncompatibleVersionException`	Valid integration point client cannot connect to server because server does not support client's version.
`NoCredentialsException`	User credentials are missing.

Item Exceptions

Item exceptions occur when there is an error processing an item while executing a command.

The following table lists the application item exceptions and the reason why they occur.

Exception Class	Reason for Exception
`InvalidItemException`	Item invalid.
`ItemNotFoundException`	Item does not exist, for example, member you are trying to drop from sandbox does not exist.

Exception Class	Reason for Exception
`ItemAlreadyExistsException`	Item is duplicate, for example, member you are trying to add to sandbox already added.
`ItemModificationException`	Item cannot be modified. For example, the Integrity Lifecycle Manager field you are trying to change is read-only.

Permission Exceptions

Permission exceptions occur when a command cannot be executed because you do not have the required permissions.

Currently, permission exceptions are only guaranteed to contain a message. They do not necessarily indicate the object or the permission being checked.

Internal Exceptions

Internal exceptions occur when there is an internal error within the integration point.

The following table lists the internal exception and the reason why it occurs.

Exception Class	Reason for Exception
`UnknownException`	Unknown error occurred while executing command.

Integrity Lifecycle Manager Client Launch Exceptions

Integrity Lifecycle Manager client launch exceptions occur when user settings prevent the Integrity Lifecycle Manager client from starting. For example, a launch exception occurs if the native path is not set to include the client installation.

Interrupted Exceptions

Interrupted exceptions occur when you try to access a response when the command runner was interrupted before the command completed.

C API Errors

The error handling approach for the C API differs from normal C error handling. Normally the *errno* global variable is used to hold the last known error code, and the `strerror()` function is used to provide an error message. To support pointers to multiple threads and ensure thread safety, the *mksrtn* variable holds the last known error code. The `mksAPIStrError()` function provides an error message.

The C AP can generate two types of errors:

- Application errors that are part of the response
- Internal API errors that are not part of the response

C API Application Errors

To find any errors that occurred as part of the response, you must search for them explicitly. For example:

```
mksrtn rtn = MKS_SUCCESS;
rtn = mksItemGetId(item, buf, sizeof(buf));
if (rtn != MKS_SUCCESS) {
/* Handle the error here. */
}
```

Some C API functions return `NULL` if an error has occurred, rather than an error code. For these functions, you must retrieve the error code in a different way.

The `mksCmdRunnerGetSession()`, `mksSessionGetIntegrationPoint()`, `mksCmdRunnerExecCmd()`, and `mksCmdRunnerExecArr` functions return pointers to `structs`. For these functions, use the following functions:

- `mksGetError()` to retrieve the last generated error
- `mksAPIStrError()` to retrieve the error message for the related error code:

An example follows:

```
mksResponse response = NULL;
response = mksCmdRunnerExecCmd(cr, cmd, NO_INTERIM);
if (response == NULL) {
rtn = mksGetError();
/* Handle the error here. */
}
```

Note

The return value of `mksGetError()` is only updated after a call is made to a function that does not return `mkstrn` or `void`. For example, calling `mksResultGetMessage()` after calling `mksResultGetField()` does not reset the return value of `mksGetError()`, which the `mksResultGetField()` function set.

For the remaining functions that return `NULL`, use the following functions:

- `mksGetError()` to retrieve the `mksrtn` error code value
- `mksAPIStrError()` to retrieve the string associated with the error

Application Error Codes

The application error codes represent only the generic application exceptions. There are many more specific exceptions that are possible within each generic application exception. Each specific exception is mapped to an exported exception name used for API exception reporting. The specific exception that occurred can be identified using the `mksAPIExceptionGetId()` method.

The following tables list the application error codes that can occur, what they mean, and the reason why they occur.

Error Codes	Description	Reason for Error
300	General application exception	Processing or validation error within Integrity Lifecycle Manager.
301	Unsupported application	Application passed in with command is unknown/invalid.
302	Internal application error	Internal error occurred on integration point while executing command.
303	No such element error	Attempted to access element that does not exist, for example, field.
310	Application connection failure	Error while initiating connection to Integrity Lifecycle Manager server.
311	Incompatible version	Valid integration point client cannot connect to server because server does not support client’s version.
312	No credentials	User credentials are missing.

Error Codes	Description	Reason for Error
320	Application runtime error	Error occurred while integration point running command because of unanticipated application problem.
321	Application out of memory error	Integration point ran out of memory while executing command.
330	Generic item exceptions	Error processing an item while executing command.
331	Invalid item	Item invalid.
332	Item exists	Item is a duplicate. For example, the member that you are trying to add to the sandbox is already added.
333	Item modification exception	Item cannot be modified. For example, the Integrity Lifecycle Manager field you that you are trying to change is read-only.
334	Item not found	Item does not exist. For example, the member that you are trying to drop from the sandbox does not exist.
340	Permission failure	Permission exceptions occur when command cannot be executed because you do not have correct permissions.

Internal Error Codes

The following tables list the internal error codes that can occur, what they mean, and the reason why they occur.

Error Code	Description	Reason for Error
200	Out of memory error	Not enough memory allocated.
201	Interrupted	Cannot access a response because command runner interrupted before command completed.
202	Integrity Lifecycle Manager client launch error	Integrity Lifecycle Manager client cannot be started because of user settings, for example,

Error Code	Description	Reason for Error
		native path not set to include client install.
203	Invalid parameter - NULL	NULL parameter passed to function expecting valid value.
204	Invalid parameter - NULL list	NULL parameter list passed to function expecting valid parameter list.
205	Insufficient buffer size	Buffer too small to accept data.
206	Invalid data type	Integration attempting to cast one data type into another incompatible one.
207	End of list	Integration attempting to access more list items that are available.
210	Command already running	Command runner already in use and cannot be used to run another command without first interrupting it.
211	Invalid command runner state	Command runner cannot be used to execute commands. For example, the command runner has been released.
220	API connection failure	Connection failure occurred when trying to connect with integration point.
221	Invalid host	API cannot connect to configured integration point. No server/client running at the configured address for integration point.
222	Invalid integration point	API can connect to address but integration point details are invalid, for example, invalid host name or invalid port.
223	Unsupported API version	API version not compatible with integration point.
230	Generic API communications error	Communication between integration and integration point broken.

Command Variations

Use the following variations when running commands:

- Using the Graphical User Interface on page 54
- Using Interim Responses on page 55
- Using Impersonation on page 56
- Using File References on page 57
- Using Unpublished Commands on page 59

Using the Graphical User Interface

If you are integrating with the Integrity Lifecycle Manager client, you can have your users interact with the client through the GUI. The GUI interaction is the same whether you use a standard session or the common session. For more information, see Sessions on page 30.

There are several ways to interact through the GUI:

- GUI for status reporting on page 54
- GUI for prompting on page 55
- Full GUI on page 55

Note

If your integration runs as an NT service and automatically starts the Integrity Lifecycle Manager client on the same system, that client is not able to perform GUI functions. This is because it has no display ability.

GUI for Status Reporting

The response for an API command gets some of its content from status information generated by the command. The status identifies the following:

- Work items that have been processed
- Any results generated from processing the work items or by the command itself
- Any errors that occurred

The status does not include informational output from view commands.

Status reporting is controlled through the `--status` command option. You can turn off status reporting `(--status=none` or you can direct status reporting to the GUI `--status=gui`.

If the status is redirected to the GUI, then all the status information is presented directly to the user through the GUI. This option is especially helpful for long running commands (for example, resyncing a sandbox) to let you know that something is happening.

If the status is redirected to the GUI, the status information is not included in the API response. The API response does, however, always contain the final exit code from the command, as well as any final exception if the command failed.

Tip

If you only need the exit code in your response, use `--status=none` to reduce the size of the response on action commands.

GUI For Prompting

GUI prompting of command settings can be added to any command run through the API by specifying the option `--settingsUI=gui`. This causes the command settings window to open before the command executes, enabling you to see the prompts and specify your settings.

When you use this option, command status information is presented to the user through the GUI. Unlike the `--status=gui` option, status information is also included in the API response.

Full GUI

The full GUI is invoked when the option `--gui` is specified on a command. Using this option redirects all information from a view command to the GUI. When you use this setting, the API response for view commands only contains an exit code and any final command exception.

Using this option also enables GUI prompting of command settings for all types of commands.

Using Interim Responses

A command can return a large amount of data or take a long time between returning the first piece of information and the last. However, you can begin reading the initial response information before the command has completed running. For example, you can use an interim response when running a query in Integrity Lifecycle Manager that returns details on thousands of items. This enables you to read the first item immediately.

If you use an interim response on a command, it returns an empty response immediately, and you can begin trying to read work items. This causes your integration program to wait until the first work item is returned from the command. Do not read command level information such as command level results, exceptions, exit codes, and so on until the command completes. Because command level information is initially unavailable, any Java API command exception is recorded in the response but not thrown when executing the command. If the work item results in an error, the appropriate exception is thrown.

Interim responses can be cached or not. Not caching improves memory management because the response is not populated with all the work items. Once you have read all of the work items, the response is empty.

Caution

If you are not caching and you request command level information, such as exit code, all of the unread work items are discarded. This occurs even when all work items have not been read.

Using Interim Responses for the Java API

- Execute the command using the `executeWithInterim ()` method.
- Start reading work items from the response using the `WorkItemIterator`.

Using Interim Responses for the C API

- Execute `mksCmdRunnerExecArr()` or `mksCmdRunnerExecCmd()` with one of the

  `INTERIM_NO_CACHE`
  or `INTERIM_CACHE` `enum` options.

Using Impersonation

In a server-to-server integration, where the server is doing work for multiple users, the integration is unable to supply passwords for every user to the Integrity Lifecycle Manager server. This is because the Integrity Lifecycle Manager server usually does not have access to them. Impersonation provides a controlled way for an integration to work on behalf of another user. Before an impersonation can be performed, an administrator must set up who can be impersonated and who is allowed to use the impersonation. For details on setting up impersonations, see Setting Up Impersonations on page 193.

You must still supply a default user and password for the impersonating user. In most cases, the default user and password are the same for all impersonated users within an integration.

Note

Each impersonated user consumes a license as if the impersonated user had logged on.

For example, if item commands are performed in the GUI as an impersonated user, their **Modified By** fields show that the impersonated user made the changes. An administrator can navigate from one view to another, displaying the impersonated user name in the views.

Java Example: Using Impersonation

```
cmdRunner.setDefaultHostname("myserver");
cmdRunner.setDefaultPort(7001);
cmdRunner.setDefaultUsername("IntegrationUser");
cmdRunner.setDefaultPassword("IntegrationUserSecret");
cmdRunner.setDefaultImpersonationUser("real user");
```

C Example: Using Impersonation

```
rtn = mksCmdRunnerSetDefaultHostname(cmdRunner, "myserver");
rtn = mksCmdRunnerSetDefaultPort(cmdRunner, 7001);
rtn = mksCmdRunnerSetDefaultUsername(cmdRunner, "IntegrationUser");
rtn = mksCmdRunnerSetDefaultPassword(cmdRunner, "Secret");
rtn = mksCmdRunnerSetDefaultImpersonationUser(cmdRunner,
"real user");
```

Using File References

Some application commands reference the file system either as an input or output, for example:

- Sandbox commands
- Check out command
- Check in command
- Commands that provide bulk data transfer
- Commands that specify an image or item attachment

These commands still interact with the file system when run through the API. There are three things to keep in mind when working with commands with file system references:

- Integration point
- Current working directory
- Remote file access

Integration Point

File system references are applied at the integration point not at the integration location. When constructing a command to send to a remote integration point, file system references must be valid on the remote system. This is because the files are read or written on the remote server. All file system references are valid when using an Integrity Lifecycle Manager client as an integration point. For security reasons, all file references on the Integrity Lifecycle Manager server are restricted to the `data/tmp` directory in the server installation directory.

If you need complete file system access but want to integrate your server application directly to an Integrity Lifecycle Manager server, see Integrating With the Integrity Lifecycle Manager Client as Server on page 16.

Current Working Directory

Several commands have implicit logic to derive settings that are not specified in the command by searching the system, starting with the current working directory. Assume that you are using the API and that no current working directory (`--cwd`) setting has been set on the command. The directory that the current integration application started in is used as the current working directory. For this reason, explicitly include all settings for files, such as the sandbox or project name. Or, determine the current working directory and pass it as the `--cwd` setting for all commands that use it.

Remote File Access

Certain commands enable you to transfer bulk data by specifying a source or output file. Assume that you are using these commands through the API when integrating with the Integrity Lifecycle Manager client. If you want to transfer bulk data to the API's file system rather than the Integrity Lifecycle Manager client's file system, you must use a special command option.

- For the Integrity Lifecycle Manager Java API, you create this option using the `com.mks.api.FileOption` class.
- For the Integrity Lifecycle Manager C API, you create this option using the `mksOptionListAddFile` method.

When you use this option, any file locations that you specify are relative to the location of the API.

For more information on Java API classes or C API methods, see the Integrity Lifecycle Manager Java API documentation or Integrity Lifecycle Manager C API documentation. Both can be accessed from the Integrity Lifecycle Manager server Web page. For more information on the bulk data commands, see the `si projectadd`, `si projectco`, `si projectci`, and `im extractattachments` commands in the CLI man pages.

Using Unpublished Commands

You can use unpublished commands to initiate commands and views that use the GUI. Much of the response content for unpublished commands is missing or incomplete. Some unpublished commands throw command level exceptions.

Unpublished command responses contain:

- The application and command
- The exit code (same as the CLI)
- Exceptions containing only a message, but no IDs or fields
- Results containing only a message

Caution

Unpublished commands used with the Integrity Lifecycle Manager API do not necessarily return predictable results. Consequently, PTC does not support unpublished commands.

For a list of published commands, see Published API Commands on page 97.

Best Practices

This section describes some of the recommended practices for using the API:

- Copying the API on page 59
- Managing Users on page 60
- Memory Management Tips on page 66
- Using Facade Patterns on page 67
- Converting Integrity Lifecycle Manager Change Management Scripts to API Programs on page 68
- Accessing Item Fields on page 68

Copying the API

If you use the API when you run your integration, do not use the same version of the API that you used to compile your integration. You must copy it and bundle with the integration.

Note

Assume that you choose to point to the `mksapi.jar` file in the Integrity Lifecycle Manager client or Integrity Lifecycle Manager server installation. By not copying and bundling the `mksapi.jar` file with your integration, you potentially introduce the following risks:

- Binary incompatibilities only detected at runtime
- Behavioral incompatibilities

When you point to the `mksapi.jar` file, you must test your integration each time the `mksapi.jar` file changes. This can occur during installation of a new version through the installer, a service pack application, or a hotfix application.

For more information on API version compatibility, see API Version Compatibility on page 77.

Note

Additionally, keep a copy of the matching Integrity Lifecycle Manager API documentation from the Integrity Lifecycle Manager server.

As an alternative to keeping a copy of the API, ensure that a newly compiled version of your integration is used with the upgraded API files.

Managing Users

An integration uses sessions differently depending on the number of users it is managing. To best understand the information in this section, familiarize yourself with the information outlined in Sessions on page 30.

Single User

A single user session is a scenario where many scripts are written for running a sequence of commands for a single user. A single user session can be used for a client or server integration point. The following examples show how to create a single user session for both integration points using Integrity Lifecycle Manager API 4.13.

Note

You specify the user and password in two places. The session is used to validate that you have authority to create a session on the server integration point. The command runner defaults identify the user to run the command as. A different user and password could be used for each of these.

Java Example: Client Integration

```
IntegrationPointFactory ipf =
IntegrationPointFactory.getInstance();
CmdRunner cmdRunner =
ipf.createLocalIntegrationPoint(APIVersion.API_4_13)
.getCommonSession()
.createCmdRunner();
// Set only to override client defaults
cmdRunner.setDefaultHostname("myserver");
cmdRunner.setDefaultPort(7001);
cmdRunner.setDefaultUsername("IntegrationUser");
cmdRunner.setDefaultPassword("secret");
```

Java Example: Server Integration

```
IntegrationPointFactory ipf =
IntegrationPointFactory.getInstance();
IntegrationPoint intPt =
ipf.createIntegrationPoint("myserver", 7001, APIVersion.API_4_13);
CmdRunner cmdRunner =
intPt.createSession("IntegrationUser","secret")
.createCmdRunner();
// Set command defaults
cmdRunner.setDefaultHostname("myserver");
cmdRunner.setDefaultPort(7001);
cmdRunner.setDefaultUsername("IntegrationUser");
cmdRunner.setDefaultPassword("secret");
```

C Example: Client Integration

```
mksIntegrationPoint ip = NULL;
mksSession session = NULL;
mksCmdRunner cr = NULL;
mksCommand cmd = NULL;
mksrtn rtn = MKS_SUCCESS;
rtn = mksAPIInitialize(NULL);
rtn = mksCreateAPIConnector(&ip, 4, 14, 1));
rtn = mksGetCommonSession(&session, ip);;
rtn = mksCreateCmdRunner(&cr, session);
/* Setup defaults for the mksCmdRunner.*/
rtn = mksCmdRunnerSetDefaultHostname(cr, "myserver");
rtn = mksCmdRunnerSetDefaultPort(cr, 7001);
rtn = mksCmdRunnerSetDefaultUsername(cr, "IntegrationUser");
rtn = mksCmdRunnerSetDefaultPassword(cr, "secret");
```

C Example: Server Integration

```
mksIntegrationPoint ip = NULL;
mksSession session = NULL;
mksCmdRunner cr = NULL;
mksCommand cmd = NULL;
mksrtn rtn = MKS_SUCCESS;
rtn = mksAPIInitialize(NULL);
rtn = mksCreateAPIConnector(&ip, "myserver", 7001, 0, 4, 14);
rtn = mksCreateSession(&session, ip, "IntegrationUser", "secret");
rtn = mksCreateCmdRunner(&cr, session);
/* Setup our defaults for the mksCmdRunner. */
rtn = mksCmdRunnerSetDefaultHostname(cr, "myserver");
rtn = mksCmdRunnerSetDefaultPort(cr, 7001);
rtn = mksCmdRunnerSetDefaultUsername(cr, "IntegrationUser");
rtn = mksCmdRunnerSetDefaultPassword(cr, "secret");
```

Multiple Users

When there are multiple users, you must ensure that one user’s actions do not have an undesirable effect on another user’s actions. You can achieve this by creating a session for each user or by sharing a session. Sharing a session allows you to manage your resources with greater efficiency.

When working with multiple users in a shared session, consider using impersonation so that the integration is not required to use each user’s password. For more information, see Using Impersonation. on page 56

The following examples show how to share a single session among multiple users. When combined with multi-threading, this allows an integration to have multiple threads, each servicing a user or multiple users sequentially.

Java Example: Sequential Users

```
IntegrationPointFactory ipf =
IntegrationPointFactory.getInstance();
IntegrationPoint intPt =
ipf.createIntegrationPoint("myserver",7001);
CmdRunner cmdRunner =
intPt.createSession("IntegrationUser","secret")
```

```
.createCmdRunner();
// Set command defaults
cmdRunner.setDefaultHostname("myserver");
cmdRunner.setDefaultPort(7001);
cmdRunner.setDefaultUsername("IntegrationUser");
cmdRunner.setDefaultPassword("secret");
// Run multiple commands
Response r = cmdRunner.execute(myCmd);
...
// Change User
cmdRunner.setDefaultUsername("AnotherUser");
cmdRunner.setDefaultPassword("theirSecret");
// Run multiple commands
r = cmdRunner.execute(myCmd);
...
```

[Repeat for as many users as necessary]

Java Example: Concurrent Users

```
// Setup a common session
IntegrationPointFactory ipf =
IntegrationPointFactory.getInstance();
IntegrationPoint intPt =
ipf.createIntegrationPoint("myserver",7001);
Session session =
intPt.createSession("IntegrationUser","secret");
...
// When changing users
// - share existing session
// - create unique cmdRunner
CmdRunner cmdRunner = session.createCmdRunner();
cmdRunner.setDefaultHostname("myserver");
cmdRunner.setDefaultPort(7001);
cmdRunner.setDefaultUsername("IntegrationUser");
cmdRunner.setDefaultPassword("secret");
cmdRunner.setDefaultImpersonationUser(myUser);
```

C Example: Sequential Users

```
mksIntegrationPoint ip = NULL;
mksSession session = NULL;
mksCmdRunner cr = NULL;
mksCommand cmd = NULL;
mksResponse response = NULL;
mksrtn rtn = MKS_SUCCESS;

rtn = mksAPIInitialize(NULL);
rtn = mksCreateAPIConnector(&ip, "myserver", 7001, 0, 4, 14);
rtn = mksCreateSession(&session, ip, "IntegrationUser", "secret");
rtn = mksCreateCmdRunner(&cr, session);
/* Setup defaults for the mksCmdRunner. */
rtn = mksCmdRunnerSetDefaultHostname(cr, "myserver");
rtn = mksCmdRunnerSetDefaultPort(cr, 7001);
rtn = mksCmdRunnerSetDefaultUsername(cr, "IntegrationUser");
rtn = mksCmdRunnerSetDefaultPassword(cr, "secret");

/* Run multiple commands */
response = mksCmdRunnerExecuteCmd(cr, myCmd, NO_INTERIM);
...
rtn = mksCmdRunnerSetDefaultUsername(cr, "AnotherUser");
rtn = mksCmdRunnerSetDefaultPassword(cr, "theirSecret");
/* Run multiple commands */
response = mksCmdRunnerExecuteCmd(cr, myCmd, NO_INTERIM);
...
```

[Repeat for as many users as necessary]

C Example: Concurrent Users

```
mksIntegrationPoint ip = NULL;
mksSession session = NULL;
mksCmdRunner cr1 = NULL;
mksCmdRunner cr2 = NULL;
mksCommand cmd = NULL;
mksResponse response = NULL;
mksrtn rtn = MKS_SUCCESS;

rtn = mksAPIInitialize(NULL);
rtn = mksCreateAPIConnector(&ip, "myserver", 7001, 0, 4, 14);
rtn = mksCreateSession(&session, ip, "IntegrationUser", "secret");
...
/* When changing users:
- share existing session
- create a unique cmdRunner
*/
rtn = mksCreateCmdRunner(&cr, session);
rtn = mksCmdRunnerSetDefaultHostname(cr1, "myserver");
rtn = mksCmdRunnerSetDefaultPort(cr1, 7001);
rtn = mksCmdRunnerSetDefaultUsername(cr1, "IntegrationUser");
rtn = mksCmdRunnerSetDefaultPassword(cr1, "secret");

/* Run multiple commands */
response = mksCmdRunnerExecuteCmd(cr1, myCmd, NO_INTERIM);
```

Dedicated Sessions

If a session is dedicated to a single user, the actions taken by one user are not affected by actions taken by another user throughout the session. If a session is used frequently, pool it for reuse, rather than destroy and re-create.

Java Example

```
// Setup integration point
IntegrationPointFactory ipf =
IntegrationPointFactory.getInstance();
IntegrationPoint intPt =
ipf.createIntegrationPoint("myserver",7001);
...
// For Each User
// - create unique session
// - create unique cmdRunner
Session session =
intPt.createSession("IntegrationUser","secret");
CmdRunner cmdRunner = session.createCmdRunner();
cmdRunner.setDefaultHostname("myserver");
cmdRunner.setDefaultPort(7001);
cmdRunner.setDefaultUsername("IntegrationUser");
cmdRunner.setDefaultPassword("secret");
cmdRunner.setDefaultImpersonationUser(myUser);
```

C Example

```
mksIntegrationPoint ip = NULL;
mksSession session = NULL;
mksCmdRunner cr1 = NULL;
mksCmdRunner cr2 = NULL;
mksCommand cmd = NULL;
mksResponse response = NULL;
mksrtn rtn = MKS_SUCCESS;

rtn = mksAPIInitialize(NULL);
rtn = mksCreateAPIConnector(&ip, "myserver", 7001, 0, 4, 14);
...
/* For each user:
- Create unique mksSession
- Create unique mksCmdRunner
*/
rtn = mksCreateSession(&session, ip, "IntegrationUser", "secret");
/* When changing users:
- share existing session
- create a unique cmdRunner
*/
```

```
rtn = mksCreateCmdRunner(&cr, session);
rtn = mksCmdRunnerSetDefaultHostname(cr1, "myserver");
rtn = mksCmdRunnerSetDefaultPort(cr1, 7001);
rtn = mksCmdRunnerSetDefaultUsername(cr1, "IntegrationUser");
rtn = mksCmdRunnerSetDefaultPassword(cr1, "secret");

/* Run multiple commands */
response = mksCmdRunnerExecuteCmd(cr1, myCmd, NO_INTERIM);
```

Memory Management Tips

You can use the following practices to manage your memory better within the integration and integration point.

Memory Management within the Integration Point

Connections consume resources, whether they are established explicitly or as part of a command pre-condition. To manage your memory effectively, consider when to disconnect a connection. If a connection is used frequently, pool its session for reuse.

In the same manner, when you are done with a session or command runner, release the integration point to free up any resources being used. Just de-referencing these objects does not free up the resources. Releasing a session or command runner does not disconnect a client from a server.

In addition, remember to execute an exit to shut down clients that are automatically restarted.

Memory Management when Releasing Elements

The C API internally maintains the API structure in a hierarchy such that an integration point contains sessions, which contain command runners (and so forth). When an element in the C API is released, all of structures it contains are released as well. For simplicity, it is recommended that you call `mksAPITerminate` at the end of your use of the C API to release all remaining structure. The `mksAPITerminate()` function releases all the cached `mksIntegrationPoint` instances and close down the logger properly.

Once a structure (and the structures it contains) is released, any handle to the structure that is being held by the integration (application) is now invalid. However, the C API cannot validate or prevent further use of these handles. It is the responsibility of the integration writer to do so.

The C API synchronization mechanism is part of the allocated structures. As a result, releasing those structures cannot be internally synchronized by the C API. In a multi-threaded integration (application), it is the responsibility of the integration writer to ensure proper synchronization in the integration itself when releasing C API structures. Specifically, ensure that the integration calls `mksAPITerminate` once on a single thread only after all other threads have completed their use of the API.

Memory Management within the Response

If a command can possibly return a large number of work items or work items with large amounts of information, consider using an interim response. For more information, see Using Interim Responses on page 55.

In the Java API, de-reference a response once it has been used so that it is available for garbage collection. Additionally, de-reference exceptions since they hold a reference to their associated response.

The C API allows an `mksResponse` instance to be released. The `mksReleaseResponse()` function releases the entire response hierarchy. Because the response contains only references to substructures, if you release it, you cannot reuse parts of the underlying hierarchy unless you copy the hierarchy.

Note

The response hierarchy of an `mksResponse` includes any of these instances: `mksSubRoutine`, `mksWorkItem`, `mksItem`, `mksField`, `mksResult`, `mksAPIException`, `mksItemList`, and `mksValueList`.

Releasing a command runner in the C API also releases any response that was generated under that command runner. As a consequence of this, a response held by an integration can no longer be safely inspected. Additionally, data cannot be extracted from it after the command runner is released. Best practice is to ensure that the following occurs:

- API response is fully read
- Important data is copied from the response into the integration before releasing the command runner

PTC also recommends that you periodically release command runners to reduce the impact on memory while the integration is in use.

Using Facade Patterns

This practice only relates to the Java API. A façade is a way of providing a simplified, standardized interface to a large set of objects. If you write an integration that involves more than running a few commands, consider isolating the Integrity Lifecycle Manager API behind a facade. This involves creating a class or classes that hide the implementation of the function you need.

For example, assume that you must know the state of an item. You can write a method such as `String getState (int issue)` to return the state as a string when given an issue as an integer. This enables you to use this function anywhere in your integration without needing to know about the workings of the API. There are times when you must make changes or go to a newer version of the API to use new features. Because your changes are isolated behind the facade, they can be independently tested.

Depending on your needs, you can create a facade for each command. Or, you can create a façade for all functions related to a single item type, such as sandboxes or items, or for all functions called.

Converting Integrity Lifecycle Manager Change Management Scripts to API Programs

When converting Integrity Lifecycle Manager change management scripts to programs using the API, you often cannot do the following:

1. Take the exact commands being run.
2. Implement them in Java or C.
3. Run them with the same results.

The reason for this is that most Integrity Lifecycle Manager change management scripts interact with the file system in some way. You must adjust the commands as described in the section Using File References on page 57.

Accessing Item Fields

Fields on an item or work item can be one of many basic data types. For example, the data type can be integer, Boolean, or string. The type can also be more complex, such as a list, item, or item list. No matter what the field data type is, you can retrieve the field as a string.

If full user name support is enabled, the user is always returned as an item. This item contains a field for the user's full name. For more information on full user name support, see the *Integrity Lifecycle Manager Installation and Upgrading Guide*. If all you want to do is determine the login ID for a user, you can do this simply with the `getValueAsString()` method.

Retrieving Item Fields as Strings for Java API

Retrieving Item Fields as Strings for the Java API

To retrieve item fields as a string for the Java API, use the `getValueAsString()` method on the Field class.

For example, run the command:

```
im issues --fields=Summary,Created By 1234
```

On getting the work item, you can determine field values:

```
String summary =
workItem.getField("Summary")
.getValueAsString();
String userlogin =
workItem.getField("Created By")
.getValueAsString();
```

For most data types, this function just returns the field value as a string.

- For an item data type, this function returns the item ID. This eliminates the requirement to get the item and then get its ID.
- For a list data type, this function returns the standard list contents available through the `java.util.List` interface.
- For an item list data type, this function returns the standard Java Foundation Class list format of the IDs of the items in the list.

Retrieving Item Fields as Strings for the C API

To retrieve item fields as strings for the C API, use the `mksFieldGetValueAsString()` function on the `Field` variable.

For example, run the command:

```
im issues --fields=Summary,Created By 1234
```

On getting the work item, you can determine the field values:

```
mksField field = NULL;
mksWorkItem wi = NULL;
wchar_t summary[1024];
wchar_t userlogin[1024];
mksrtn rtn;
...
field = mksWorkItemGetField(wi, L"Summary");
rtn = mksFieldGetValueAsString(field, summary, sizeof(summary));
...
field = mksWorkItemGetField(wi, L"Created By");
rtn = mksFieldGetValueAsString(field, userlogin, sizeof(userlogin));
```

For most data types, this function just returns the field value as a string `(wchar_t *)`.

- For an `mksItem` value, this function returns the ID.
- For an `mksItemList` value, this function returns a comma-separated list of IDs.

- For an `mksValueList` value, this function returns a comma-separated list of the elements.
- For a `NULL` value, this function returns an error code of `MKS_NULL_VALUE`.

Extracting Images in Workflow and Document Admin Objects

From the Integrity Lifecycle Manager Java API, you can extract images in workflow and document admin objects (such as types, queries, and fields), allowing you to use the images. For example, using a Java application, you can extract images to create admin objects on a different Integrity Lifecycle Manager server. Or, you can create additional admin objects when the original images are no longer available.

For all admin objects, image data is presented in the API as item data. A value for an item or image field is provided in the response for each admin view. The item data contains an id, description, and optional width, height, and data fields (the data field contains the binary representation of the image).

For example:

`Item:`

- `Id` = `none`, `default`, or `custom`.
  - `none` and `default` contain the `description` field.
  - `custom` contains the following fields: `description, width, height`, and `data`. The `data` field is a binary field from which the actual image data can be retrieved. You can use the `byte[] getBytes()` method to return the raw image data, exported as a PNG image file. Or, you can use the `ImageIcon getImage()` method to return a Java `ImageIcon`, ready for display.
- `Model Type` = `im.Image`. This is defined as `IMAGE` in `com.mks.api.im.IMModelTypeName`.

Key Considerations

Extracting images is supported only from the Java API, with the images extracted using a Java application. Extracting images from the CLI or C API is unsupported.

- The following admin objects containing an image support image extraction:
  - users
  - groups
  - dynamic groups

  - query
  - states
  - types
- The following admin objects containing additional images support image extraction:
  - projects
  - state overrides
  - phase field phases
  - pick field picks
  - range field ranges
  - relationship field flags

Dashboards and item presentation templates do not support image extraction. The extracted image file type is always PNG.

The following sample Java code illustrates how to extract an image from the `myState` workflow and document Integrity Lifecycle Manager field:

```
ImageIcon icon = null;
try {
    WorkItem state = apiResponse.getWorkItem("myState");
    Item imageItem = state.getField("image").getItem();
    if (imageItem.contains("data")) {
          Field imageData = imageItem.getField("data");
          icon = imageData.getImage();
    }
}
// you can then proceed to show or save image
```

Viewing Input Requirement Information Using Constraints Commands

The Integrity Lifecycle Manager API includes constraints commands that provide the ability to view information about all of the input requirements for creating or editing Integrity Lifecycle Manager resources. This includes both system-defined limitations and administrator-created constraints and definitions.

This functionality enables you to create external integrations and apps that are consistent with existing Integrity Lifecycle Manager input requirements. By using the information gathered using a constraints command, you can create an intelligent user interface that conforms to these input requirements.

Each constraints command responds with the constraints for a named type of Integrity Lifecycle Manager object. For example, the `im itemconstraints` command returns constraint shapes that describe the input requirements for fields on Integrity Lifecycle Manager items when creating or editing that item. The constraints that are returned are specifically applicable to the current Integrity Lifecycle Manager configuration context and the user who is making the API request.

Retrieving Constraints

The API response for constraints command uses a fixed structure to describe input requirements. This structure is inspired by Shapes Constraint Language (SHACL). It follows the layout of an Integrity Lifecycle Manager view command response.

The work items in each response represent a *constraint shapes container*, each containing two fields:

- `scopeItem`—The item to which constraint shapes apply
- `shapes`—An item list containing a single closed constraint shape

The closed constraint shape that is provided in the response contains an `OperandShapes` field. This field lists all of the constraint shape descriptors that apply to the item. A *closed constraint shape* simply reflects that the union of the constraints described within it represent the complete set of all fields relevant to Integrity Lifecycle Manager input requirements. Any field that is not mentioned is not relevant for Integrity Lifecycle Manager input.

Each constraint shape has a `Constraints` field, which is an item list of property constraint descriptors. Each property constraint describes the constraint applied to a specific field. It has the following structure:

API Field Name	Description	SHACL Equivalent
`AttributeName`	The name of the field on the item to which this constraint applies	`sh:predicate`
`DataType`	The expected data type for values supplied to this field	`sh:datatype`
`ModelType`	Indicates that the field expects values that are references to other items of the named model type. Mutually exclusive with `DataType`.	`sh:valueClass`
`ValueKind`	The format of the values for the field:	`sh:nodeKind`

API Field Name	Description	SHACL Equivalent
	• Literal—Simple data value • Reference—ID reference to another Integrity Lifecycle Manager item • Structure—Input data requires a structure with multiple fields	
`ValueShape`	A constraint shape that describes the fields required for the structure of value objects that can be set on this field	`sh:valueShape`
`ReadOnly`	Indicates that this field is not editable. If not specified, the field is editable.	
`MinimumCount`	The minimum number of values that can be set on this field. If not specified, the field has a minimum count of 0, which means that the field can be empty. If the minimum count is higher than 0, the field is required.	`sh:minCount`
`MaximumCount`	The maximum number of values that can be set on this field. If not specified, the field has an infinite maximum count, which means that the field is multi-valued. A maximum count of 1 means that the field is single-valued.	`sh:maxCount`
`MinInclusive` `MaxInclusive`	For fields that accept a range of data values, these API fields indicate	`sh:minInclusive` `sh:maxInclusive`

API Field Name	Description	SHACL Equivalent
	the minimum and maximum values in that range	
`MaximumLength`	For fields that accept string values, this API field indicates the maximum number of characters allowed for input values	`sh:maxLength`
`DefaultValue`	The default values that Integrity Lifecycle Manager normally applies to this field when creating the associated item. This API field is only applicable when viewing constraints for the creation of new items.	`sh:defaultValue`
`AllowedValues`	For fields where values are constrained to a specific set of values, this API field indicates the list of values that are allowed	`sh:in`
`Suggestions`	For fields where Integrity Lifecycle Manager recommends preferred values, but the field is not constrained to only these values, this API field indicates the list of suggestions	

With each property constraint or constraint shape, a `FilterShape` field may exist. This field references a constraint shape that describes a filter condition. This filter condition must be matched before the property constraint or constraint shape applies.

The filter condition can be one of the following:

- A core constraint that represents one of the following:
    - AND, OR, or NOT conditions

    - A pair constraint that represents the comparison between fields
- A property constraint that represents the evaluation of a field against a constant

Each filter is provided to integration consumers as expressions so that integration consumers can apply their own conditional evaluations.

Using Constraints Commands

For more information about using constraint commands, see Constraints Commands on page 187.

Creating Programs for the C API

For all platforms, `mksapi.h` is the main include file, and `#include <mksapi.h>` includes all the files that you must compile.

Windows (Visual C++)

Add:

- `<`*`Integrity Client`*`>/include>` as an additional include directory.
- `<`*`Integrity Client`*`>/bin` as an additional library directory
- `mksapi.lib` as an additional library to link

Solaris

You need:

- `-IIntegrity Client/include` as a compiler flag
- `-LIntegrity Client/lib -lmksapi -lapiblimp -lxml2 -lssl -lnspr4 -lcrypto -lpthread -lm -lsocket -lnsl` as linker flags

Linux

- `-I<Integrity Client>/include` as a compiler flag
- `-LIntegrity Client/lib -lmksapi -lapiblimp -lxml2 -lssl -lnspr4 -lcrypto -lpthread -lz -lm` as linker flags

4

API Version Compatibility

The Integrity Lifecycle Manager API includes support for version compatibility, allowing you to write an integration that can successfully continue operating across multiple versions of Integrity Lifecycle Manager. However, this requires an understanding of what aspects of the API are contractual and protected by this compatibility statement. These aspects differ from other aspects of API use that are informational and flexible.

To ensure compatibility with older integrations as you upgrade Integrity Lifecycle Manager, the following aspects of the Integrity Lifecycle Manager API are contractual:

- Command syntax (token, options, and syntactic interpretation of values).

  Note

  Command defaults are not contractual. However, there is some support to assist older integrations in accommodating the addition of new command options of which they would not have been aware. Additional policies and permissions that can further affect commands are not contractual. Older integrations can legitimately fail with new modes of failure due to policy changes. It is the Integrity Lifecycle Manager administrator’s responsibility to install policy configurations that allow older and existing integrations to continue operating.

- Command primary intent (behavior). Secondary side-effects are not contractual.

- Overall API response pattern for published commands. This does not apply to commands that are not published. For more information on published API commands, see Published API Commands on page 97
- API field tokens, especially where these are related to a command field setting (as opposed to those related to the Integrity Lifecycle Manager data model), are constants.
- Model Type tokens for API items. These are contractual because they allow an integration to identify what type of Integrity Lifecycle Manager domain object the API item represents.
- The data type for a given API field. The actual value; however, is not contractual because it is based on the Integrity Lifecycle Manager data model.
- API exception classification (regarding the mapping and tokenization of Integrity Lifecycle Manager errors and exceptions). However, it is not contractual that an integration necessarily receives the same exception for the same command under the same condition. New exception conditions could be introduced (or restructured) as Integrity Lifecycle Manager changes.

This chapter describes these aspects of the Integrity Lifecycle Manager API, in addition to providing some best practices in using the API.

- Understanding Integrity Lifecycle Manager API Versions on page 79
- Command Execution on page 89
- Command Response on page 80

Understanding Integrity Lifecycle Manager API Versions

The API provides a framework to invoke Integrity Lifecycle Manager commands and receive responses. Command compatibility via the API is constrained by the commands that are supported for the API and by the command changes, if any, that affect compatibility.

When supporting an integration across multiple versions of Integrity Lifecycle Manager, it is important to understand the following:

- Integrity Lifecycle Manager API version used by your integration
- Integrity Lifecycle Manager API versions used by the versions of Integrity Lifecycle Manager communicating with your integration

The API version reported by Integrity Lifecycle Manager is the highest API version that it can support. For example, Integrity 10.4 reports Integrity Lifecycle Manager API version 4.12. This means that this is the highest version it supports when an integration communicates with it. However, it also supports API sessions for 4.11 and 4.10 when requested.

Note

Compatibility is only ensured by the integration builder explicitly requesting an API response for a given version. The API version is a version number that belongs to the integration.

For more information on specific API versions, see API Versions on page 24.

Updates to Integrity Lifecycle Manager API

Every attempt is made to ensure that changes to the Integrity Lifecycle Manager API minimize the impact on existing integration programs written to the API. However, there is no contract that the Integrity Lifecycle Manager API's object structure or methods are explicitly maintained. Any update to the API is essentially an update to the integration code. The expectation is that to update the API within an integration is a deliberate undertaking and results in recompiling, retesting, and revalidating the integration.

The benefits that the Integrity Lifecycle Manager API offers include the following:

- Ability to retain and use a single version of the API (module)
- Ability to use this single version of the API to communicate with multiple (upgraded) versions of Integrity Lifecycle Manager

Command Response

In addition to communicating command invocations to the Integrity Lifecycle Manager integration point, the Integrity Lifecycle Manager API also collects and reports the command's response to the integration.

This section describes the compatibility aspects of the command's API response, focusing on Integrity Lifecycle Manager views (commands used to retrieve data from Integrity Lifecycle Manager).

Published Command Responses

The hallmark of a published API command, aside from the installation of API appropriate defaults, is the API response. Published API responses are classified as follows:

1. Creational commands produce an API response that contains an exception (in the event of failure), or an API result (in the event of success). The primary value of the result is an API item, representing the object created by the command. This item can contain additional informational fields. For example:

```
…
try {
    Response response = cmdRunner.execute(creationCmd);
    Item createdItem = response.getResult().getPrimaryValue();
    …
}
catch (APIException creationFailure) {
   …
}
```

2. Operational commands perform an implicit or explicit action on a selection of Integrity Lifecycle Manager domain objects. The API response from a successful command execution contains a collection of API work items. These work items represent the selection of objects on which the command can operate. Each work item contains one of the following:
   - An exception if the operation on the given item fails
   - An API result with the primary value representing the result if the operation on the given item succeeds

```
…
Response response = null;
try {
   response = cmdRunner.execute(operationCmd);
}
catch (APIException ex) {
   // Check to see if the exception is a command-fatal exception.
   …
```

```
       // Otherwise it just an indication that at least one of the
       // items operated on contains a failure.
       // Obtain the response for further evaluation as such …
       response = ex.getResponse();
    }
    …
    if (response != null) {
        WorkItemIterator selection = response.getWorkItems();
        while (selection.hasNext()) {
            try {
                Item selectionItem = selection.next();
                Item resultItem =
                      selectionItem.getResult().getPrimaryValue();
                …
            }
            catch (APIException failure) {
                // Thrown by selection.next() in the event that
                // there was an exception in operating on the
                // selection item.  To determine the item that
                // the failure occurred on …
                Item failedSelectionItem = selection.getLast();
                …
            }
        }
    }
```

3. Retrieval commands are used to obtain information from Integrity Lifecycle Manager. The API response for these commands contains a collection of API work items, representing the items retrieved. Each work item contains a collection of API fields, representing the requested information on the given item. Each API field consists of a field name/token and a value where the value can be one of the following:
   - A simple value, such as a text string or date
   - An API item representing a complex object value, which in turn can have fields
   - A collection of simple values or API items

```
    …
    try {
        WorkItemIterator items =
              cmdRunner.execute(retrievalCmd).getWorkItems();
        while (items.hasNext()) {
            Item retrievedItem = items.next();
            Iterator fields = retrievedItem.getFields();
            while (fields.hasNext()) {
                Field retrievedField = (Field) fields.next();
                …
            }
        }
    }
    catch (APIException ex) {
        // Exceptions on retrieval commands are rare
        // unless there is some system or policy failure
        …
    }
```

In publishing a command, the nature of the command and its corresponding API response pattern is determined and then becomes fixed. Integrations can rely on Integrity Lifecycle Manager to ensure that the nature and general format of the API response remains the same as Integrity Lifecycle Manager is upgraded.

For published commands, the Integrity Lifecycle Manager API offers all integrations the following:

- There is no need to parse messages to retrieve information. All useful information is presented in the API as an explicit field value or item data.
- In the majority of cases, the name of an API field is a token constant that, once created for a published command, does not change. In the remaining few cases, the fields and their names are obtained from Integrity Lifecycle Manager's domain data model. This is only applicable to retrieval informational fields. For example, when retrieving the fields on an Integrity Lifecycle Manager item, the `Workflow` type definition for this item defines the field names.

Some retrieval commands accept a `--fields` option. The token name of the fields requested in the command invocation matches the token name of the fields in the API response.

- Each item contains an ID (`item.getId()`) which, in a published command, is expected to be a token representation of the item. In general, this same ID is applicable, without change, for use in a subsequent command invocation when referring to the item of interest.
- Each item has a model type (`item.getModelType()`) which, in a published command, is a token constant that does not change. However, new items with new model types can be introduced in newer Integrity Lifecycle Manager versions.

Note

Integrity Lifecycle Manager uses the API request version to ensure that the API response pattern and its tokens and ID representations are contractual and remain available for integrations. This does not mean that the total command output is restored to its former state when an older API request version is received. There is allowance for the API response to expand as Integrity Lifecycle Manager is upgraded. Any information that a well written integration could rely on when first implemented can be relied on later. This assumes usage of the correct API request version and availability of the same information.

Command Exit Code

Ultimately, the Integrity Lifecycle Manager API communicates with Integrity Lifecycle Manager to execute commands. Command execution, aside from the API request version, inherits the CLI command compatibility considerations, as offered and implemented by Integrity Lifecycle Manager.

Basic command compatibility and CLI best practices include:

- Command tokens and command options are preserved (no command token or option is removed between Integrity Lifecycle Manager releases).

  This means that the command syntax used in one release remains valid in a future release. However, while the command syntax remains the same, the outcome and behavior of the command could change.

- With any change in command behavior, there is a corresponding (possibly new) command option that can control this behavior.

  With a newer Integrity Lifecycle Manager version, the behavior of previous versions of the command can effectively be restored with the proper use of (possibly new) command options.

- With every command option, there is a corresponding command preference.

  This means that command options that are unspecified in the command invocation use a configured default preference to control command behavior. This preference can be taken from built-in defaults within Integrity Lifecycle Manager or from a solution/customer configuration.

  In the CLI, compatibility for commands can be restored to the behavior of previous versions of Integrity Lifecycle Manager by installing appropriate custom preferences for the (new) command options.

In the API, using the request version, Integrity Lifecycle Manager automatically applies appropriate preferences for the (new) command options. This effectively restores the operations in newer versions of the command, based on invocations from the requested version.

- Command defaults are not contractual and are subject to change.

  Command defaults, as based on the preferences discussed previously, are already subject to change within a given release of Integrity Lifecycle Manager, due to administrative and user configuration. Versions of Integrity Lifecycle Manager can include different built-in defaults, as per the intent of the release.

  CLI best practices recommend that where certain command operations are essential, you explicitly use the appropriate command options. Do not rely on the command defaults at the time of implementation.

The same CLI best practices apply when using the Integrity Lifecycle Manager API.

Ordering

When discussing API responses, the term collection is used often. The collection of items and collection of fields is an ordered collection in the API. However, this is merely a convenience. The actual order is not contractual.

The convention is that when a command accepts an input selection of items to operate on, the output follows the same order as the input. This is a convention and not a contract. In a given version of Integrity Lifecycle Manager, some commands follow this convention and some others do not follow them. This can further change between release versions.

The API request version does not have any influence over how a command orders its output. Order is actually an artifact of how the command is implemented and therefore subject to change as the implementation changes.

Changes to Values

Specifically considering the values in API fields, three ways exist for retrieving a field value:

1. String representation (`field.getValueAsString()`).
2. Untyped object value (`field.getValue()`).
3. Typed object value, based on the field's reported data type.

The actual value is part of the data domain and is considered informational.

The string representation retrieval for a value is useful if the value is simply being reported or displayed. However, it does offer the advantage of always being available and most responsive to changes.

Retrieving the value as an untyped object is equally guaranteed to always be available. However, you would typically have to then cast the object to a type to perform a useful action with it. Similar to the final retrieval method, this assumes that the integration can expect or anticipate the data type for the value. As such, the data type for the value of a constant token field should be contractual and under the control of the API request version. Newer versions of Integrity Lifecycle Manager can change the data type for a value. However, they equally have an obligation to convert it to its original data type when a given API request version is received.

Note

- While there are many cases where Integrity Lifecycle Manager implements contractual compatibility support for API values, there are also a number of cases where this has not been implemented. Many of these cases are areas in which the original data type was considered to have been incorrect. The contract is not upheld to retain “bug-wise” compatibility with older integrations. However, this is under reconsideration.
- Consider a case where an integration expects a value to be of a specific data type and this contract is not met. The type-specific value retrieval method (such as `field.getBoolean()`) can throw a runtime exception `(java.lang.UnsupportedOperationException).`
- Consider a case where API values are not thought to be correct and are changed without contractual compatibility support. This can occur when the initial output for a field is a string value and then later refined to be a stronger-typed value. In such cases, integrations can mitigate the potential impact of such changes by retrieving the value as a string representation (`field.getValueAsString()`), instead of the controlled means to retrieve the value specifically as a string object (`field.getString()`).

Handling Additional Items, Model-Types, and Fields

With the Integrity Lifecycle Manager API, you must explicitly request data from the API response, for example, `response.getWorkItem("foo"); item.getField("bar")`, or iterate over the same API response, for example, `response.getWorkItems(); item.getFields()`. As a result, Integrity

Lifecycle Manager assumes that it can add new items (using different model types) and/or add new fields to an API response without impacting existing integrations.

Contractually, Integrity Lifecycle Manager does not remove fields and more as expected with an older integration (as specified through an API request version). However, it is possible new fields and more are added to the API response.

Recommended Best Practices

PTC recommends implementing integrations with the expectation that they can retrieve more data than anticipated. However, this is not a concern if the integration explicitly and directly uses the Integrity Lifecycle Manager API to request specific items and fields. In this case, the integration does not make the request for any of the additional items or fields, resulting in no impact.

If the integration iterates over collections of items and fields, then the possibility of encountering unexpected items must be allowed. This in turn can result in model types that are unknown and/or unexpected to the integration. One implementation pattern is to validate that you are dealing with an item that the integration understands, skipping those that it does not. For example:

```
…
Iter at or items = itemList.getItems();
while (items.hasNext()) {
    Item item = (Item) items.next();
    if (item.getModelType().equals(IMModelTypeName.ISSUE)) {
        // Integration expects ISSUE items,
        // process them here
        …
   }
    else {
        // Unknown/unexpected item.  Skip it.
        continue;
    }
    …
}
```

Similarly, you can configure the integration to process only known/expected fields.

Message Handling

The Integrity Lifecycle Manager API can display messages in API results and exception messages, and can be displayed to users or captured in logs. They are modeled after the messages in Integrity Lifecycle Manager and are subject to change. These messages are information data elements, not contractual. The API request version has no impact on these messages and no attempt is made to preserve them for any existing integration.

This follows the premise that you cannot acquire useful data for published commands by parsing output messages. Integrations are expected to adhere to this premise.

Recommended Best Practices for Message Handling

PTC recommends that integrations never rely on messages for programmatic purposes. For a published API command, programmatic information should be available from explicit item or field values.

If you use a command that is not published and must rely on a message for processing, the integration should accommodate the risk and possibility that the message can change between release versions or even hotfixes.

Exception Handling

Exception handling in the Integrity Lifecycle Manager API can be a difficult contract to describe and program for; however, this is under review for future Integrity Lifecycle Manager API releases.

First, the Integrity Lifecycle Manager API has a hierarchy of exceptions that reflect the inability of the API to successfully communicate with Integrity Lifecycle Manager and execute commands. This is not within the scope of any compatibility contract, since it does not really involve Integrity Lifecycle Manager. This is simply part of the programming model presented by the implemented/installed instance of the Integrity Lifecycle Manager API (it nothing to do with the API request version).

Second, exceptions are sometimes used as programming markers to indicate the overall success or failure of a command. In many cases, when an integration must become involved with programmatic details, these are not the exceptions of interest. The discussion on various API response command patterns suggests how you acknowledge these markers and proceed with retrieving the details, including the actual exception of interest.

Finally, there is a hierarchy of exceptions under the API's `ApplicationException` class. The Integrity Lifecycle Manager API attempts to reflect the errors and exceptions generated within the processing of commands in Integrity Lifecycle Manager as an `ApplicationException` error message.

In this case, the role of the Integrity Lifecycle Manager API is to accurately reflect the exception of interest from Integrity Lifecycle Manager. The contract is with the form as to how the application exception is represented. However, there is no contract as to which specific exception an integration receives from Integrity Lifecycle Manager. As mentioned previously, Integrity Lifecycle Manager can possibly introduce new policies and conditions, leading to new exception cases. Changes to the implementation within Integrity Lifecycle Manager can also change the order in which conditions are met or validated. In either case, it is possible for an integration to receive a different exception running the same command against an upgraded version of Integrity Lifecycle Manager. This occurs because, programmatically, the Integrity Lifecycle Manager API can report on the first exception encountered only.

Recommended Best Practices for Exception Handling

PTC recommends that integrations always be prepared to allow for and handle exceptions when running a command or inspecting its response. Additionally, PTC recommends that integrations be prepared to further inspect the exception for known conditions and, if necessary, handle unexpected or unknown exceptions.

The manner in which an exception is described through the Integrity Lifecycle Manager API is considered contractual. The following are the two primary means of describing the application exception:

1. Each application error/exception is mapped onto a specific API exception class. At a broad level, this allows for an integration to respond to a classification of exceptions.

   Common exception classifications include:

   - `InvalidItemException`—Indicates an invalid item specification in the invocation.
   - `ItemNotFoundException`—Indicates a problem in locating an existing item.
   - `ItemAlreadyExistsException`—Indicates an attempt to (re)create an existing item.
   - `ItemModificationException`—Indicates any failure to update an item.
   - `PermissionException`—Indicates that a user does not have permissions to execute the operation.

   Note

   In this context, the term item is a generic reference to any Integrity Lifecycle Manager domain object; it is not limited to Integrity Lifecycle Manager workflow items.

2. Each application exception in the Integrity Lifecycle Manager API also includes an Integrity Lifecycle Manager application exception ID/token (a `piException.getExceptionId()`). This is an additional tokenized

data point from Integrity Lifecycle Manager that an integration can use to further classify the nature of the application exception.

Note

Exception ID/tokens are expected to be treated as constants (that an integration can rely on), making them contractual. However, there is no contract that prevents an integration from receiving an alternate exception if conditions change.

The mapping of Integrity Lifecycle Manager errors and exceptions to both an API exception class and the tokenization/identification of the type of Integrity Lifecycle Manager errors/exceptions are contractual, as indicated earlier. Consequently, they are influenced by the API request version. If the mapping or tokenization changes in an Integrity Lifecycle Manager release, Integrity Lifecycle Manager has an obligation to respect an API request version. Integrity Lifecycle Manager is required to restore the mapping or tokenization to the form an older integration expects.

PTC recommends that when an integration must make programmatic decisions based on an exception, the integration bases the decisionon the specific API exception class being produced. Then, if necessary, the tokenized exception ID that Integrity Lifecycle Manager provides can be used.

Note

Never allow an integration to make a programmatic decision based on an exception message. Exception messages are informational only. They are intended to be used for display or logging. Exception messages are subject to change between releases of Integrity Lifecycle Manager.

Command Execution

Ultimately, the Integrity Lifecycle Manager API communicates with Integrity Lifecycle Manager to execute commands. Command execution, aside from the API request version, inherits the CLI command compatibility considerations, as offered and implemented by Integrity Lifecycle Manager.

Basic command compatibility and CLI best practices include:

- Command tokens and command options are preserved (no command token or option is removed between Integrity Lifecycle Manager releases).

This means that the command syntax used in one release remains valid in a future release. However, while the command syntax remains the same, the outcome and behavior of the command can change.

- With any change in command behavior, there is a corresponding (possibly new) command option that can control this behavior.

  Consequently, with a newer Integrity Lifecycle Manager version, the behavior of previous versions of the command can effectively be restored with the proper use of (possibly new) command options.
- With every command option, there is a corresponding command preference.

  This means that command options that are unspecified in the command invocation use a configured default preference to control command behavior. This preference can be taken from built-in defaults within Integrity Lifecycle Manager or from a solution/customer configuration.

  In the CLI, compatibility for commands can be restored to the behavior of previous versions of Integrity Lifecycle Manager by installing appropriate custom preferences for the (new) command options.

  In the API, using the request version, Integrity Lifecycle Manager automatically applies appropriate preferences for the (new) command options. This effectively restores the operations in newer versions of the command, based on invocations from the requested version.
- Command defaults are not contractual and are subject to change.

  Command defaults, as based on the preferences discussed previously, are already subject to change within a given release of Integrity Lifecycle Manager, due to administrative and user configuration. Versions of Integrity Lifecycle Manager can include different built-in defaults, as per the intent of the release.

  CLI best practices recommend that where certain command operations are essential, explicitly use the appropriate command options. Do not rely on command defaults at the time of implementation.

The same CLI best practices apply when using the Integrity Lifecycle Manager API.

Command Syntax

As indicated earlier, the Integrity Lifecycle Manager API inherits the basic command compatibility that Integrity Lifecycle Manager offers. This means that any command syntax (command tokens and options) used in implementing integrations for a given version of Integrity Lifecycle Manager remain valid for newer versions of Integrity Lifecycle Manager. The command syntax is contractual.

To further preserve tokens that refer to commands and command options, as already supported in the CLI compatibility, the API allows Integrity Lifecycle Manager to use the request version. The request version retains compatible syntactical interpretation of command option values and selections.

Note

Integrity Lifecycle Manager, with its heavy use in CLI scripting, already maintains syntactic compatibility for command option and selection values, independent of any API request version.

There is no specific recommendation for command syntax. Integrity Lifecycle Manager is responsible for all compatibility support for command syntax. Integrations can assume such support.

Note

Newer versions of Integrity Lifecycle Manager can deprecate command options, so they do not need to be documented or appear in the command usage. However, these options are only hidden. They remain available for use by older scripts and integrations.

Command Defaults

A default value is associated with every command option. This default helps to inform the command on how to operate if the command option is not explicitly specified.

Command defaults, even within a single release version of Integrity Lifecycle Manager, are always subject to change. Defaults can be changed by data, environment, user configuration, administrative configuration, or changes to the version of Integrity Lifecycle Manager. They are not contractual.

For example, consider what fields the following command outputs:

```
im issues --query=Defects
```

In earlier versions, Integrity Lifecycle Manager would output a fixed set of fields, or at least it would seem so. However, the user or administrator could change this using advanced command preference configurations. In later Integrity Lifecycle Manager versions, defaults for the output fields were based on a configuration stored with the query. The query determined the set of fields outputted. The defaults were subject to change.

Relying on command defaults can be useful because it allows an integration to delegate some control and choice to Integrity Lifecycle Manager and administrative configuration. In these cases, the integration must be written with the expectation that the command output and behavior are likely to change.

Recommended Best Practices for Command Defaults

When an integration requires specific command behavior, PTC recommends explicitly specifying the appropriate command option. Do not rely on command defaults.

For example, the following command explicitly ensures that the **ID** and **Summary** fields are output used by the integration:

```
im issues --query=Defects --fields=ID,Summary
```

New options can be added to a command in an updated version of Integrity Lifecycle Manager. The new options are missing from command invocations in existing integrations. This means that the existing integrations are forced to accept the command defaults for the new options. In the case of the Integrity Lifecycle Manager API, missing command options are handled in the following ways:

- Unlike the CLI, the Integrity Lifecycle Manager API is explicitly non-interactive. In the case where a new option is added without clear defaults, in the CLI, this can result in new user interaction asking for input. With the Integrity Lifecycle Manager API, this results in a command execution error because an answer cannot be provided for the new option.

  However, this error condition, is a fallback position that should only occur with the use of unpublished API commands. When using unpublished commands, integrations must be developed with the possibility that invocations can fail. For published commands, additional information follows.
- In many cases, new command options are added to Integrity Lifecycle Manager, along with new command defaults. The defaults effectively allow for the command to operate as it would before the addition of the option. This is the basis for Integrity Lifecycle Manager CLI script compatibility.

  In some cases, the notion of compatibility can differ between the CLI and API use. In such cases, Integrity Lifecycle Manager is able to install defaults that are different for the API command execution versus the CLI command execution. For example, consider the introduction of display patterns to certain workflow fields. The command to retrieve data on an item defaults to displaying the fields with the patterns applied in the CLI. This is because the data is primarily used for user reporting. In the API execution of the same command, defaults are not applied. This is because the raw value is generally of more interest to integrations.

- Consider cases where new command options are either deliberately left without options or default to confirm with the user during CLI user interaction. Integrity Lifecycle Manager installs separate API defaults for published commands that best allow for the command to safely proceed.

  For example, the CLI asks you to whether to overwrite a local file when extracting an attachment from an Integrity Lifecycle Manager item. The API default for the same command simply overwrites the local file, allowing for the command to proceed as requested.
- Although the Integrity Lifecycle Manager API and CLI can have different defaults for command execution, Integrity Lifecycle Manager supports default overrides based on API request versions. This allows Integrity Lifecycle Manager to support the API compatibility necessary to allow older integrations to continue working as designed.

Note

Because command defaults are not contractual, the use of the API request version to influence command defaults is reserved. The API request version influences command defaults for cases only when it significantly and materially affects the API response and integrations. For example, consider the introduction of non-exclusive locking. The use of non-exclusive locks in Integrity Lifecycle Manager and new integrations was encouraged. However, integrations based on the 4.8 API request version were defaulted to stay with exclusive locks. This was because of the major change in how locks would be reported if they were non-exclusive.

Recommended Best Practices

PTC recommends that you do not use the `--yes/-Y` or `--no/-N` option when invoking commands in scripts or through the Integrity Lifecycle Manager API. Instead, explicitly specify each relevant confirmer option in the command invocation.

Automatically answering unknown confirmations with a generic yes or no answer should have little impact on using the Integrity Lifecycle Manager API with published commands. However, PTC does not consider doing this a good practice. Automating the response can seem convenient when dealing with a familiar command from a version of Integrity Lifecycle Manager in a given scenario. However, a change in either the scenario or the version can inadvertently result in new or unexpected confirmation questions being asked. This can lead to the command not operating as expected.

Command Execution Policies and Preferences

When developing integrations, it is important to distinguish between policies and preferences.

Preferences are the configuration used to implement command defaults and are associated with command options, as described earlier. There is no contract that preferences are preserved between Integrity Lifecycle Manager versions. However, given good practices and defaults for published API command, there is reasonable support to ensure that integrations continue to operate successfully as Integrity Lifecycle Manager is upgraded. Good practices include explicitly specifying required command options when implementing an integration. Administrators and users can also generally configure ore reconfigure the Integrity Lifecycle Manager installation to ensure that preferences reflect the needs of any pre-existing integration.

Policies are generally associated with the Integrity Lifecycle Manager and its environment. Policies can affect command operations, but they are generally not controlled by command options. New policies can be implemented in upgraded versions of Integrity Lifecycle Manager based on strategic reasons, business decisions, and best practices from the industry. Examples include the new association of existing operations with mandatory change packages and the addition of new permissions for restricting certain operations.

Unfortunately, the imposition of new/unexpected policies can result in command execution failure. There is little an integration can do about this by itself, except be prepared to gracefully handle command failure.

An administrator can change policies and permissions within an installation. In some cases (and as a best practice), a new version of Integrity Lifecycle Manager installs new policies and permissions. If these affect existing integrations, the administrator can change the policies and permissions to allow for an existing integration to continue to operate. Otherwise, the integration must be updated to use new command options or new commands to work with the new policies.

Command Behavior and Potential Side-Effects

When developing integrations, it is important to distinguish between a command’s primary intent and outcome and any potential side-effects that can result from using the command.

In some cases, potential side-effects can result from the solutions employed at a specific installation. A solution includes the combination of rules, policies, triggers, and automated scripts that respond to command invocations as well changes to data. In this case, the integration must be viewed as part of the entire solution. Changes to any part of the solution can affect an integration. However, this is outside of the scope of any compatibility support that the Integrity Lifecycle Manager API can offer.

In other cases, side-effects can occur based on an implementation of Integrity Lifecycle Manager and the user interface (UI). For example, when creating or editing an item in the Integrity Lifecycle Manager UI, a change to one field can result in a change to another field. However this is a side-effect of the UI. This side-effect should not be expected to occur when the same create or edit is performed by an integration through the Integrity Lifecycle Manager API. In this case, the integration is expected to explicitly provide values to all relevant fields to allow the create or edit to proceed as expected.

In other cases, side-effects are merely secondary operations or results of a command. In these cases, what is perceived as a side-effect can merely be a response to different scenarios. For example, the common result of checking in a new revision, based on member revision at 1.x, is the creation of revision 1.(x+1). However, this operation could also result in the creation of 1.x.1.1 or 1.(x+2). The creation of a new revision is the primary outcome. However, the actual revision number created is the secondary outcome and based on environmental scenarios. In cases like this, PTC recommends that you do not make too many assumptions. Instead, rely on the command response to inform the integration of the actual outcome.

There are many cases where side-effects, or what appears to be side-effects, are subject to change. PTC recommends exercising caution. In some cases, side-effects can be mitigated with explicit use of command options. In other cases, they are outside of the direct control of the command invocation. However, the integration can be implemented in a manner prepared to handle the effect of the command, based on its response.

A command’s primary intent, however, is not expected to change. This is considered contractual.

Assume that a command’s intentional behavior changes in an upgraded version of Integrity Lifecycle Manager. There is an obligation to ensure that the same command reverts to its original intentional behavior when the appropriate API request version is specified. This means that Integrity Lifecycle Manager ensures that command intent is preserved for existing integrations.

Note

Command intent is loosely embodied in a documented statement as to what the command is expected to accomplish. This does not override the impact to the command operations and behavior that can be imposed with the introduction of new command options, defaults, and policies. For example, by definition, a checkin is always expected to create a new revision. This assumes that all policies and command options are met. Policies include mandatory change packages, and command options include whether to ignore unresolved merges. Whether a checkin also updates member revision to the checked in revision is a side-effect based on command options, defaults, and scenarios.

5

Published API Commands

Every command and command option on the Integrity Lifecycle Manager and Integrity Lifecycle Manager server can be called through the Integrity Lifecycle Manager API. However, PTC supports only published commands for use with the Integrity Lifecycle Manager API.

A published command is one that meets the following criteria:

- Uses an established and appropriate API response pattern. For more information, see Command Response Summary on page 80
- Contains an API response where data is appropriately divided and matched (ready for input) to other published commands. Data output from a published view (command) should be usable, where appropriate, as input to a separate published command.
- Contains an API response that is consistent with a requested API version.

Unpublished commands used with the Integrity Lifecycle Manager API do not always return predictable results. Consequently, PTC does not support them. For more information, see Using Unpublished Commands on page 59.

For information on the commands and their options, see the CLI man pages.

For information on the custom change package commands and their options, see the *Integrity Lifecycle Manager Installation and Upgrading Guide*.

Note

The terms item and issue refer to the same Integrity Lifecycle Manager object and are indistinguishable. Issue is a term embedded in legacy command and option names; therefore, item and issue are used interchangeably in the API and CLI documentation

When you are designing integrations, PTC strongly recommends that you use the API Viewer to become familiar with the expected output for API commands. For more information, see Viewing Responses With the API Viewer Utility on page 41.

This chapter contains the following information:

Format for Command Information

Command information in this chapter is provided in the following format.

Command

Describes the command, including the command type (action, view, creation). For more information on command types, see Reading Responses on page 35.

Work Item

Describes the item that the command worked on, such as a member.

Work Item Context

Describes the context of the work item, for example, the project for the member.

Work Item Result

Describes the result of the work item.

For action commands, lists any potential fields for command results.

For view commands, lists any fields that are specifically added to the view by the API. Also lists fields which are required to maintain structure on exposed field values. All other fields exposed for view commands are documented in the man pages for the commands and in the CLI man pages.

PTC strongly recommends that you use the API Viewer to become familiar with the expected output for API commands. For more information, see Viewing Responses With the API Viewer Utility on page 41.

Command Result

Describes the result of the command.

Result Context

Describes the context of the result, for example, the change package type for the change package attribute.

Note

No explicit order is implied by the sequence of the entries in any of the tables.

Published Commands

Published commands include the following information:

- Workflow and Document Management on page 100
- Configuration Management on page 145
- Authorization Administration on page 179
- Integrity Lifecycle Manager server on page 182
- Working File on page 183

Workflow and Document Management Commands

All published Workflow and Document Management commands start with the command prefix `im`.

im about

about

`im about`

This view command can be used within the API to return product information.

Work Item

The application for which the product information is returned.

im baseline

baseline

`im baseline`

This command labels (or relabels) the indicated segment roots, optionally as of a specified timestamp.

Work Item

The segment selection.

Work Item Result

The labeled segment.

im branchsegment

branchsegment

`im branchsegment`

This command branches one or more segments by branching the segment root and all the nodes in the structural hierarchy. All fields listed in the `copyFields` type attribute are copied to the new item.

The selection must be either a segment or a node that references a subsegment.

Work Item

The segment selection.

Note

When the `--swap` option is specified, the command reports the node as the work item and indicates which segment was swapped out. The subsegment reference type stays the same.

Work Item Result

The branched segment.

im changesegmentproject

changesegmentproject

`im changesegmentproject`

This command changes the project in the segment. When the `--subsegment` option is used and the selection is a node that references a subsegment, then the subsegment's project is changed. Otherwise, the project is changed for the segment containing the selected node.

Work Item

The segment selection.

Work Item Result

List of all subsegments whose projects have been changed.

im charts

charts

`im charts`

This view command can be used within the API to return a list of Integrity Lifecycle Manager charts (created or shared).

Work Item

The charts for which the information is returned.

Work Item Context

The names of the users that created the charts.

Work Item Fields

The following table lists the fields added by the API for the charts. These properties are editable from the CLI using the `im editchart` command. For information on all available fields, see the CLI man pages

Field	Data Type	Description
`chartType`	String	Type of chart.
`createdBy`	User	Name of the user who created the chart.
`description`	String	Description of the chart.
`graphStyle`	String	Graph style used by the chart.
`id`	Integer	Database ID of the chart. This is for PTC Technical Support only.
`isAdmin`	Boolean	Specifies the chart as a system-provided object (objects within the Integrity Lifecycle Manager object model that support solution definition and management, as well as workflow migration).
`lastModified`	Date	Date the chart was last modified.
`name`	String	Name of the chart, and the user who created the

Field	Data Type	Description
		chart.
`query`	Query	Name of the query that defines the selection criteria for the chart, and the user who created the chart.
`references`	List of objects	All system provided and user objects that reference the chart.
`shareWith`	List of users and groups	Users and groups that the chart is shared with.
`sharedGroups`	List of groups	Groups that the chart is shared with.

im ci

ci

`im ci`

This view command can be used within the API to return a list of Integrity Lifecycle Manager charts (created or shared).

Work Item

All newly created versioned items.

Work Item Result

The model type (segment or content) and the display ID.

im connect

connect

`im connect`

This action command can be used within the API to establish a connection to an Integrity Lifecycle Manager server.

im copycontent

copycontent

`im copycontent`

Use this action command to copy a selection of nodes (and optionally all nodes underneath them).

Note

By default, each node is explicitly branched from its original node. Specify the `--no branch` option to copy the node without a branch record being created.

Work Item

The content selection.

Work Item Result

The ID of the original item and the ID of the newly created item.

im copyissue

copyissue

`im copyissue`

This creation command can be used within the API to create a new Integrity Lifecycle Manager item by copying the common fields of an existing item. The list of item types available in your database varies. The list depends on the types created by your administrator and the item types that your workflow permits you to submit.

Command Result

Specifies the item ID created by the command.

im cpattributes

cpattribute

`im cpattribute`

This view command can be used within the API to return a summary of Integrity Lifecycle Manager change package attributes for a change package type. For information on these commands and their options, see the CLI man pages.

Work Item

The change package attribute for which the information is returned.

Work Item Context

The change package type.

Work Item Result

The fields that show for each work item are dependent on the change package type and the attribute data type. For example, a `Logical` field does not have any `Strings` values.

The following table lists the fields added by the API for change package attributes. For information on all available fields, see the CLI man pages.

Field	Data Type	Description
`Name`	String	Real name of the change package attribute.
`DisplayName`	String	Name displayed to users.
`Description`	String	Detailed description of the attribute.
`Type`	String	Data type of the change package attribute.
`Position`	Integer	Position of the attribute in the change package attribute list.
`IsReadOnly`	Boolean	Specifies if the value of the attribute is read-only.
`IsMandatory`	Boolean	Specifies if a value for the attribute is mandatory.
`MaxLength`	Integer	Maximum length of the change package attribute value.
`DecimalPlaces`	Integer	Number of decimal places allowed in change package attribute value.
`DisplayFormat`	String	Format in which the attribute value displays.
`ID`	Integer	Identification number of the change package attribute.
`Strings`	List of strings	A list of possible values.

im cpentryattributes

cpentryattributes

`im cpentryattributes`

This view command can be used within the API to return a summary of Integrity Lifecycle Manager change package entry attributes for a change package type. For information on these commands and their options, see the CLI man pages.

Work Item

The change package entry attribute for which the information is returned.

Work Item Context

The change package type.

Work Item Result

The fields that show for each work item are dependent on the change package type and the attribute data type. For example, a `Logical` field does not have any `Strings` values.

The following table lists the fields added by the API for change package entry attributes. For information on all available fields, see the CLI man pages.

Field	Data Type	Description
`Name`	String	Real name of the change package attribute.
`DisplayName`	String	Name displayed to users.
`Description`	String	Detailed description of the attribute.
`Type`	String	Data type of the change package attribute.
`Position`	Integer	Position of the attribute in the change package attribute list.
`IsReadOnly`	Boolean	Specifies if the value of the attribute is read-only.
`IsMandatory`	Boolean	Specifies if a value for the attribute is mandatory.
`MaxLength`	Integer	Maximum length of the change package attribute value.
`DecimalPlaces`	Integer	Number of decimal places allowed in change package attribute value.
`DisplayFormat`	String	Format in which the attribute value displays.
`ID`	Integer	Identification number of the change package attribute.
`Strings`	List of strings	A list of possible values.

im cps

cps

`im cps`

This view command can be used within the API to return a summary of Integrity Lifecycle Manager change packages.

Work Item

The change package attribute for which the information is returned.

Work Item Fields

In addition to the fields specified by the `--attributes` and `--entryAttributes` options, the API adds the following field for organizing the list of entry items for the change package being viewed.

For information on all available fields, see the CLI man pages.

Field	Data Type	Description
`MKSEntries`	List of entry items	List of entries in change package. Only available for `viewcp` command. For information on the change package entry fields, see the CLI man pages.

im cptypes

cptypes

`im cptypes`

This view command can be used within the API to return a summary of Integrity Lifecycle Manager change packages.

For information on these commands and their options, see the CLI man pages.

Work Item

The change package attribute for which the information is returned.

Work Item Result

For information on all available fields, see the CLI man pages.

Field	Data Type	Description
`ID`	Integer	Identification number of type.
`Name`	String	Real name of the change package type.
`DisplayName`	String	Name displayed to users.
`Position`	Integer	Order number for type.
`Description`	String	Detailed description of the change package type.

Field	Data Type	Description
`Attributes`	Change package attribute item	List of change package attributes associated with the change package type. For information on the change package attribute fields, see the *Integrity Lifecycle Manager Installation and Upgrading Guide*.
`EntryAttributes`	Change package attribute items	List of change package entry attributes associated with the change package type. For information on the change package entry attribute fields, see the *Integrity Lifecycle Manager Installation and Upgrading Guide*.
`EntryKey`	Change package entry attribute items	List of change package entry attributes to be used as a key (to uniquely identify change package entries). For information on the change package entry attribute fields, see the *Integrity Lifecycle Manager Installation and Upgrading Guide*.
`PermittedGroups`	List of groups	Groups allowed to use this type.
`PermittedAdministrators`	List of users or groups	Users or groups allowed to perform administration operations on type.

im creategroup

creategroup

`im creategroup`

This creation command creates a new group.

For information on this command and its options, see the CLI man pages.

Command Result

Specifies the group that was created.

im createchart

createchart

`im createchart`

This creation command creates a new chart.

For information on this command and its options, see the CLI man pages.

Command Result

Specifies the chart that was created.

Result Context

The user who created the chart.

im createcontent

createcontent

`im createcontent`

This creation command creates a new node and shared item couplet, and inserts the node under the specified parent.

For information on this command and its options, see the CLI man pages.

Command Result

New content item is created.

im createcp

createcp

`im createcp`

This creation command can be used within the API to create an Integrity Lifecycle Manager change package.

For information on this command and its options, see the CLI man pages.

Command Result

Specifies the change package ID that was created.

im createcpattribute

createcpattribute

`im createcpattribute`

Use this creation command within the API to create an Integrity Lifecycle Manager change package attribute for a change package type.

For information on this command and its options, see the CLI man pages.

Command Result

Specifies the change package attribute that was created.

Result Context

The change package type.

im createcpentry

createcpentry

`im createcpentry`

This creation command can be used within the API to create an Integrity Lifecycle Manager change package entry.

For information on this command and its options, see the CLI man pages.

Command Result

Specifies the change package ID for which the entry was created.

Result Context

The change package type.

im createcpentryattribute

createcpentryattribute

`im createcpentryattribute`

This creation command can be used within the API to create an Integrity Lifecycle Manager change package entry attribute for a change package type.

For information on these commands and their options, see the CLI man pages

Command Result

Specifies the change package entry attribute that was created.

Result Context

The change package type.

im createcptype

createcptype

`im createcptype`

This creation command can be used within the API to create a new Integrity Lifecycle Manager change package type. Common change package attributes are automatically created.

For information on this command and its options, see the CLI man pages.

Command Result

Specifies the change package type created.

im createdashboard

createdashboard

`im createdashboard`

This creation command creates a new dashboard.

For information on this command and its options, see the CLI man pages.

Command Result

Specifies the dashboard that was created.

Result Context

The user who created the dashboard.

im createdynamicgroup

createdynamicgroup

`im createdynamicgroup`

This creation command creates a new dynamic group.

For information on this command and its options, see the CLI man pages.

Command Result

Specifies the dynamic group that was created.

im createfield

createfield

`im createfield`

This creation command creates a new field.

For information on this command and its options, see the CLI man pages.

Command Result

Specifies the field that was created.

im creategroup

creategroup

`im creategroup`

This creation command creates a new group.

For information on this command and its options, see the CLI man pages.

Command Result

Specifies the group that was created.

im createissue

createissue

`im createissue`

This creation command can be used within the API to create an Integrity Lifecycle Manager item. The list of item types available in your database varies. The list depends on the types created by your administrator and what item type your workflow permits you to submit.

Command Result

Specifies the item ID number that was created.

im createproject

createproject

`im createproject`

This creation command creates a new workflows and documents project.

For information on this command and its options, see the CLI man pages.

Command Result

Specifies the project that was created.

im createquery

createquery

`im createquery`

This creation command creates a new query.

For information on this command and its options, see the CLI man pages.

Command Result

Specifies the query that was created.

Result Context

The user who created the query.

im createreport

createreport

`im createreport`

This creation command creates a new report.

For information on this command and its options, see the CLI man pages.

Command Result

Specifies the report that was created.

Result Context

The user who created the report.

im createsegment

createsegment

`im createsegment`

This creation command creates a new segment root. Optionally it inserts the segment root as a subsegment underneath a node or other segment root.

Command Result

Specifies a new segment root.

im createstate

createstate

`im createstate`

This creation command creates a new state.

For information on this command and its options, see the CLI man pages.

Command Result

Specifies the state that was created.

im createtrigger

createtrigger

`im createtrigger`

This creation command creates a new event trigger.

For information on this command and its options, see the CLI man pages.

Command Result

Specifies the event trigger that was created.

im createtype

createtype

`im createtype`

This creation command creates a new item type.

For information on this command and its options, see the CLI man pages.

Command Result

Specifies the item type that was created.

im createuser

createuser

`im createuser`

This creation command creates a new user.

For information on this command and its options, see the CLI man pages.

Command Result

Specifies the user that was created.

im dashboards

dashboards

`im dashboards`

Use this view command within the API to return a list of Integrity Lifecycle Manager dashboards (created or shared) or the properties of a specific dashboard. You can specify which fields in the dashboards are displayed. If you do not specify any fields for a list of dashboards, only the names of the created and shared dashboards are returned.

Work Item

The dashboards for which the information is returned.

Work Item Context

The name of the user who created the dashboard.

Work Item Fields

The following table lists the fields added by the API for the dashboards. These properties are editable from the CLI using the `im editdashboard` command. For information on all available fields, see the CLI man pages.

Field	Data Type	Description
`createdBy`	User	Name of the user who created the dashboard.
`created`	Date	Date the dashboard was created.
`description`	String	Description of the dashboard.
`fieldFilterConstraint`	String	Stores the project filter used for the dashboard. Applies only for the `im viewdashboard` command.
`id`	Integer	Database ID of the dashboard. This is for PTC Technical Support only.
`isAdmin`	Boolean	Specifies the dashboard as a system-provided object (objects within the Integrity Lifecycle Manager object model that support solution definition and management, as well as workflow migration).

Field	Data Type	Description
`lastModified`	Date	Date the dashboard was last modified.
`layout`	String	XML representation of the dashboard layout. Layout must conform to a specified format. For more information, see the *Integrity Lifecycle Manager Help Center*.
`modifiedBy`	User	Name of the user who last modified the dashboard.
`name`	String	Name of the dashboard, and the user who created the dashboard.
`shareWith`	List of users and groups	Users and groups with whom the dashboard is shared.
`sharedGroups`	List of groups.	Groups with whom the dashboard is shared.

im deletecp

deletecp

`im deletecp`

This action command can be used within the API to delete an Integrity Lifecycle Manager change package.

Command Result

Specifies the change package ID that was deleted.

im deletecpattribute

deletecpattribute

`im deletecpattribute`

This action command can be used within the API to delete one or more Integrity Lifecycle Manager change package attributes for a change package type.

For information on this command and its options, see the CLI man pages.

Command Result

Specifies the change package ID that was deleted.

Result Content

The change package type.

im deletecpentryattribute

deletecpentryattribute

`im deletecpentryattribute`

This action command can be used within the API to delete one or more Integrity Lifecycle Manager change package entry attributes from a change package type.

For information on this command and its options, see the CLI man pages.

Command Result

Specifies the change package entry attribute from which the entry was deleted.

Result Context

The change package type.

im deletecptype

deletecptype

`im deletecptype`

This action command can be used within the API to delete one or more Integrity Lifecycle Manager change package types.

For information on this command and its options, see the CLI man pages.

Command Result

Specifies the change package type deleted.

im diffsegments

diffsegments

`im diffsegments`

This view command can be used with the API to view the differences between the following:

- One document at two different points in time
- Two versions of the same document
- Two documents that share a common branch ancestor ID

Work Item

The type of difference identified by the document comparison, which is one of the following:

- `Identical`
- `Modify`
- `Move`
- `Move and Modify`
- `Delete`
- `Add`

Following the type of difference is the row location of the difference within the source and target documents in the **Document Difference** view (`leftposition:rightposition`).

Work Item Fields

The following table lists the fields added by the API for items for the `im diffsegments` command.

Field	Data Type	Description
`ID`	Integer	Item ID.
`asOf`	Date	Date the item was compared at.
`position`	Integer	Row location of the item.
`sourceItem`	Item	Source item ID being differenced. Identifies the item ID of the parent item, the previous sibling item (`sibling.before`), and the next sibling item (`sibling.after`). Identifies the model type (segment, content, inserted subsegment, or included subsegment) and the display ID.

Field	Data Type	Description
`targetItem`	Item	Target item ID being differenced. Identifies the item ID of the parent item, the previous sibling item (`sibling.before`), and the next sibling item (`sibling.after`). Identifies the model type (segment, content, or inserted subsegment, or included subsegment) and the display ID.
`Changed Fields`	Item List	Specific fields on items that have changed.

im disconnect

disconnect

`im disconnect`

This action command can be used within the API to disconnect from the Integrity Lifecycle Manager server.

Command Result

Specifies the Integrity Lifecycle Manager server that was disconnected.

im editchart

editchart

`im editchart`

This action command can be used within the API to make changes to a chart.

For more information on this command and its options, see the CLI man pages.

Work Item

The chart that is to be updated.

Work Item Context

The user who created the chart.

Work Item Result

The updated chart.

im editcpattribute

editcpattribute

`im editcpattribute`

This action command can be used within the API to make changes to an Integrity Lifecycle Manager change package attribute.

For more information on this command and its options, see the CLI man pages.

Work Item

The change package attribute that must be updated.

Work Item Context

The change package type.

Work Item Result

Specifies the change package attribute that was updated.

im editcpentry

editcpentry

`im editcpentry`

This action command can be used within the API to make changes to an Integrity Lifecycle Manager change package entry.

For more information on this command and its options, see the CLI man pages.

Work Item

The change package ID that is to be updated.

Work Item Result

Specifies the change package ID that was updated.

im editcpentryattribute

editcpentryattribute

`im editcpentryattribute`

This action command can be used within the API to make changes to an Integrity Lifecycle Manager change package entry attribute.

For more information on this command and its options, see the CLI man pages.

Work Item

The change package entry attribute that is to be updated.

Work Item Context

The change package type.

Work Item Result

Specifies the change package entry attribute that was updated.

im editcptype

editcptype

`im editcptype`

This action command can be used within the API to make changes to an Integrity Lifecycle Manager change package type.

For more information on this command and its options, see the CLI man pages.

Work Item

The change package type to be updated.

Work Item Result

Specifies the change package type that was updated.

im editdashboard

editdashboard

`im editdashboard`

This action command can be used within the API to make changes to a dashboard.

For more information on this command and its options, see the CLI man pages.

Work Item

The dashboard to be updated.

Work Item Context

The user who created the dashboard.

Work Item Result

The dashboard that is updated.

im editdynamicgroup

editdynamicgroup

`im editdynamicgroup`

This action command can be used within the API to make changes to an Integrity Lifecycle Manager dynamic group.

For more information on this command and its options, see the CLI man pages.

Work Item

The dynamic group that is to be updated.

Work Item Result

Specifies the dynamic group that was updated.

im editfield

editfield

`im editfield`

This action command can be used within the API to make changes to a field.

For more information on this command and its options, see the CLI man pages.

Work Item

The field that is to be updated.

Work Item Result

The field that was updated.

im editissue

editissue

`im editissue`

This action command can be used within the API to make changes an Integrity Lifecycle Manager item.

Note

All mandatory fields must be specified as required when the command is executed because this command does not try to make the change again if there is a problem.

Work Item

The item that is to be updated.

Work Item Result

Specifies the item that was updated.

im editgroup

editgroup

`im editgroup`

This action command can be used within the API to make changes to a group.

For more information on this command and its options, see the CLI man pages.

Work Item

The group to be updated.

Work Item Result

The group that was updated.

im editproject

editproject

`im editproject`

This action command can be used within the API to make changes to a workflows and documents project.

For information on this command and its options, see the CLI man pages.

Work Item

The project that is to be updated.

Work Item Result

The project that was updated.

im editquery

editquery

`im editquery`

This action command can be used within the API to make changes to a query.

For information on this command and its options, see the CLI man pages.

Work Item

The query to be updated.

Work Item Context

The user who created the query.

Work Item Result

The updated query.

im editreport

editreport

`im editreport`

This action command can be used within the API to make changes to a report.

For information on this command and its options, see the CLI man pages.

Work Item

The report to be updated.

Work Item Context

The user who created the report.

Work Item Result

The updated report.

im editstate

editstate

`im editstate`

This action command can be used within the API to make changes to a state.

For information on this command and its options, see the CLI man pages.

Work Item

The state to be updated.

Work Item Result

The updated state.

im edittrigger

edittrigger

`im edittrigger`

This action command can be used within the API to make changes to an event trigger.

For information on this command and its options, see the CLI man pages.

Work Item

The event trigger to be updated.

Work Item Result

The updated event trigger.

im edittype

edittype

`im edittype`

This action command can be used within the API to change an item type.

For more information on this command and its options, see the CLI man pages.

Work Item

The item type to be updated.

Work Item Result

The updated item type.

im edituser

edituser

`im edituser`

This action command can be used within the API to make changes to an event trigger.

For information on this command and its options, see the CLI man pages.

Work Item

The user who is to be updated

Work Item Result

The updated user.

im extractattachments

extractattachments

`im extractattachments`

This action command can be used within the API to save one or more attachments from an Integrity Lifecycle Manager item.

Work Item

The filename of the attachment.

Work Item Context

The Integrity Lifecycle Manager item to which the attachment belongs.

im exit

exit

`im exit`

This action command can be used within the API to exit the current Integrity Lifecycle Manager client session.

im fields

fields

`im fields`

This view command can be used within the API to return a list of fields. By default, all fields are selected to be returned.

Work Item

The field for which the information is returned.

Work Item Fields

The fields that show for each work item depend on the field type. For example, a string field does not have `Min` or `Max` values. For more information on the data types for fields, see the *Integrity Lifecycle Manager Installation and Upgrading Guide*.

im groups

groups

`im groups`

This view command can be used within the API to return a list of groups. By default, all groups are selected to be returned.

Work Item

The group for which the information is returned.

im importissue

importissue

`im importissue`

This action command can be used within the API to import an item into the Integrity Lifecycle Manager database, creating a new Integrity Lifecycle Manager item to represent it.

Command Result

Specifies the item ID that was imported into Integrity Lifecycle Manager.

im insertsegment

insertsegment

`im insertsegment`

This action command inserts an existing segment under a node (or another segment root), either as an included subsegment or reference subsegment.

Work Item

The segment selection.

Work Item Result

Inserted segment.

im issues

issues

`im issues`

This view command can be used within the API to return a list of Integrity Lifecycle Manager items.

Work Item

The item for which the information is returned.

Work Item Fields

The following table lists the fields added by the API for items for the `viewissue` command. For information on all available fields, see the CLI man pages.

Field	Data Type	Description
`MKSIssueHistory`	List of issue change items	Historical changes made to the item.
`MKSIssueAnnotations`	List of annotation items	Annotations for the item.
`MKSIssueLabels`	List of label items	Labels on the item.
`MKSIssueBranches`	List of issue items	Items branched from the item.
`MKSIssueParent`	Integer	Item from which the current item is branched.

Field	Data Type	Description
`MKSIssueChangePackages`	List of change package items	Change packages associated with the item.
`MKSIssueTimeEntries`	List of time entries	Time entries associated with the item. For more information on the item time entries fields, see the CLI man pages.

Note

Integrity 10.5 and later includes functionality that allows you to create document and content versions. When new versions are created, the versioned items display with a unique version ID `(LiveID-Major Revision.minor` revision).

You can construct the version ID using the following fields:

`Live Item ID`

`Major Version ID`

`Minor Version ID`.

For example, a version ID of 576-1.0.

Certain commands also accept the version ID as input through the command line interface.

For more information on document versioning, see the *Integrity Lifecycle Manager Installation and Upgrading Guide* and the CLI man pages.

im lock

lock

`im lock`

This command locks a document for editing.

For information on this command and its options, see the CLI man pages.

Work Item

The segment selection.

Work Item Result

The locked segment.

im movecontent

movecontent

`im movecontent`

This action command moves the selected node from its current location to a new location, potentially to a different document. The order the selections are specified is the order in which they appear under their new parent.

If `--norecurse` is specified and a selection is a section root, its former contents are moved up to the old parent in the same location and order in which they appeared under the selection. Then, the selection is moved to its new location without any children.

If `--recurse` is specified and a selection is a section root, the children are moved with the selection.

If `--recurse` is specified and one of the selections is already included as a child or grandchild of one of the other selections, the selection is removed from the list and not reported as a work item in the API output.

Work Item

The content selection.

Work Item Result

The new parent, location of siblings and exact model type.

im projects

projects

`im projects`

This view command can be used within the API to return a list of projects. By default, all projects are selected to be returned.

Work Item

The project for with the information is returned.

Work Item Fields

The following table lists the field added by the API for projects. For information on all available fields and options, see the CLI man pages.

Field	Data Type	Description
`InheritedPermittedGroups`	Boolean	Specifies if project inherits permitted groups from its parent. If this field is set to true, then `PermittedGroups` field is empty. To find permitted groups, you have to look at parent (and possibly its parent).

im queries

queries

`im queries`

This view command can be used within the API to return a list of Integrity Lifecycle Manager queries. By default, all queries that are currently shared with you are returned.

Work Item

The query for which the information is returned.

Work Item Context

The name of the user who created the query.

im refmode

refmode

`im refmode`

This action command changes the reference mode for document nodes to either share or reuse.

Work Item

The document selection.

Work Item Result

The document reference mode is changed.

im relationships

relationships

`im relationships`

This view command displays and expands the relationships for the specified items.

Caution

The default API behavior for the `im relationships` command expands all forward relationships. Assume that you are view relationships from a root item, such as a backing project item. If you do not specify any command options, the API can return a very large dataset of items. Before using the `im relationships` command with the API, ensure that you understand the nature of the specified items used with the command. Also ensure that you specify the appropriate options to return specific results. The primary use of the `im relationships` command is to explicitly expand known relationships for a specific item.

Work Item

The items specified with the command and related items returned by the command.

Work Item Result

Fields specified by the `im relationships --fields` option. For information on all available fields and options, see the CLI man pages.

im removebaseline

removebaseline

`im removebaseline`

This action command removes the label from the indicated segments.

Work Item

The specified document.

Work Item Result

Label removed from specified document.

im removecontent

removecontent

`im removecontent`

This action command unlinks one or more nodes from the document in which they are contained. You can make selections from different segments.

Work Item

The specified content.

Work Item Result

Content is no longer linked to the document.

im reports

reports

`im reports`

This view command can be used within the API to return a list of created or shared Integrity Lifecycle Manager reports. It can also be used to return the properties of a specific report. You can specify which fields in the reports display. If you do not specify any fields for a list of reports, only the names of created and shared reports are returned. If you do not specify any fields for a specific report, all fields display.

Work Item

The reports for which the information is returned.

Work Item Context

The name of the user who created the report.

Work Item Fields

The following table lists the fields added by the API for the reports. These properties are editable from the CLI using the `im editreport` command. For information on all available fields, see the CLI man pages.

Field	Data Type	Description
`createdBy`	User	Name of the user who created the report.
`description`	String	Description of the report.
`id`	Integer	Database ID of the report. This is for PTC Technical Support only.
`isAdmin`	Boolean	Specifies the report as a system-provided

Field	Data Type	Description
		object (objects within the Integrity Lifecycle Manager object model that support solution definition and management, as well as workflow migration).
`lastModified`	Date	Date the report was last modified.
`name`	String	Name of the report, and the user who created the report.
`query`	Query	Name of the query that defines the selection criteria for the report, and the user who created the query.
`references`	List of objects	All system provided and user objects that reference the report.
`shareWith`	List of users and groups	Users and groups that the report is shared with.
`sharedGroups`	List of groups.	Groups that the report is shared with.
`reportTemplate`	String	Report template on which the report is based. For information on the report template format, see the *Integrity Lifecycle Manager Installation and Upgrading Guide*.
`recipeParams`	String	Report recipe parameters that define the report. For more information on the available parameters, see the knowledge base in the Support section of the PTC Web site (http://www.ptc.com).

im runtrigger

runtrigger

`im runtrigger`

This action command runs an event trigger on the Integrity Lifecycle Manager server.

Work Item

The event trigger that runs on the Integrity Lifecycle Manager server.

Work Item Result

The event trigger that runs on the Integrity Lifecycle Manager server.

When run, event triggers can also return parameters. As a result, the API response for the `runtrigger` command includes return parameters as fields within the work item result.

im servers

servers

`im servers`

You can use this view command within the API to return the current connections to an Integrity Lifecycle Manager server.

Work Item

The server connection for which the information is returned.

im states

states

`im states`

You can use this view command within the API to return a list of item states in Integrity Lifecycle Manager. By default, all states are listed.

Work Item

The item state for which the information is returned.

im types

types

`im types`

You can use this view command within the API to return a list of Integrity Lifecycle Manager item types. By default, all types are returned.

Work Item

The item type for which the information is returned.

im unlock

unlock

`im unlock`

This command unlocks a locked document. For information on this command and its options, see the CLI man pages.

Work Item

The segment selection.

Work Item Result

The unlocked segment.

im users

users

`im users`

You can use this view command within the API to display a list of users. By default, all users are displayed.

Work Item

The user for which the information is returned.

im viewcp

view

`im viewcp`

This view command can be used within the API to return a summary of Integrity Lifecycle Manager change packages.

Work Item

The change package attribute for which the information is returned.

Work Item Fields

In addition to the fields specified by the `--attributes` and `--entryAttributes` options, the API adds the following field for organizing the list of entry items for the change package being viewed.

For information on all available fields, see the CLI man pages.

Field	Data Type	Description
`MKSEntries`	List of entry items	List of entries in change package. Only available for `viewcp` command. For information on the change package entry fields, see the CLI man pages.

im viewcpattribute

viewcpattribute

`im viewcpattribute`

This view command can be used within the API to return a summary of Integrity Lifecycle Manager change package attributes for a change package type. For information on these commands and their options, see the CLI man pages.

Work Item

The change package attribute for which the information is returned.

Work Item Context

The change package type.

Work Item Result

The fields that show for each work item are dependent on the change package type and the attribute data type. For example, a `Logical` field does not have any `Strings` values.

The following table lists the fields added by the API for change package attributes. For information on all available fields, see the CLI man pages.

Field	Data Type	Description
`Name`	String	Real name of the change package attribute.
`DisplayName`	String	Name displayed to users.
`Description`	String	Detailed description of the attribute.
`Type`	String	Data type of the change package attribute.
`Position`	Integer	Position of the attribute in the change package attribute list.
`IsReadOnly`	Boolean	Specifies if the value of the attribute is read-only.
`IsMandatory`	Boolean	Specifies if a value for the attribute

Field	Data Type	Description
		is mandatory.
`MaxLength`	Integer	Maximum length of the change package attribute value.
`DecimalPlaces`	Integer	Number of decimal places allowed in change package attribute value.
`DisplayFormat`	String	Format in which the attribute value displays.
`ID`	Integer	Identification number of the change package attribute.
`Strings`	List of strings	A list of possible values.

im viewcpentryattributes

viewcpentryattributes

`im viewcpentryattributes`

This view command can be used within the API to return a summary of Integrity Lifecycle Manager change package entry attributes for a change package type. For information on these commands and their options, see the CLI man pages.

Work Item

The change package entry attribute for which the information is returned.

Work Item Context

The change package type.

Work Item Result

The fields that show for each work item are dependent on the change package type and the attribute data type. For example, a `Logical` field does not have any `Strings` values.

The following table lists the fields added by the API for change package entry attributes. For information on all available fields, see the CLI man pages.

Field	Data Type	Description
`Name`	String	Real name of the change package attribute.
`DisplayName`	String	Name displayed to users.
`Description`	String	Detailed description of the attribute.
`Type`	String	Data type of the change package attribute.

Field	Data Type	Description
`Position`	Integer	Position of the attribute in the change package attribute list.
`IsReadOnly`	Boolean	Specifies if the value of the attribute is read-only.
`IsMandatory`	Boolean	Specifies if a value for the attribute is mandatory.
`MaxLength`	Integer	Maximum length of the change package attribute value.
`DecimalPlaces`	Integer	Number of decimal places allowed in change package attribute value.
`DisplayFormat`	String	Format in which the attribute value displays.
`ID`	Integer	Identification number of the change package attribute.
`Strings`	List of strings	A list of possible values.

im viewcptype

viewcptype

`im viewcptype`

This view command can be used within the API to return detailed information on a selection of change package type.

For more information, see `im cptypes` on page 107.

im viewdashboard

viewdashboard

`im viewdashboard`

This view command can be used within the API to return a list of Integrity Lifecycle Manager dashboards (created or shared) or the properties of a specific dashboard. You can specify which fields in the dashboards are displayed. If you do not specify any fields for a list of dashboards, only the names of the created and shared dashboards are returned.

Work Item

The dashboards for which the information is returned.

Work Item Context

The name of the user who created the dashboard.

Work Item Fields

The following table lists the fields added by the API for the dashboards. These properties are editable from the CLI using the `im editdashboard` command. For information on all available fields, see the CLI man pages.

Field	Data Type	Description
`createdBy`	User	Name of the user who created the dashboard.
`created`	Date	Date the dashboard was created.
`description`	String	Description of the dashboard.
`fieldFilterConstraint`	String	Stores the project filter used for the dashboard. Applies only for the `im viewdashboard` command.
`id`	Integer	Database ID of the dashboard. This is for PTC Technical Support only.
`isAdmin`	Boolean	Specifies the dashboard as a system-provided object (objects within the Integrity Lifecycle Manager object model that support solution definition and management, as well as workflow migration).
`lastModified`	Date	Date the dashboard was last modified.
`layout`	String	XML representation of the dashboard layout. Layout must conform to a specified format. For more information, see the *Integrity Lifecycle Manager Help Center*.
`modifiedBy`	User	Displays the name of the user who last modified the dashboard.
`name`	String	Name of the dashboard, and the user who created the dashboard.
`shareWith`	List of users and groups	Users and groups with whom the dashboard is shared.
`sharedGroups`	List of groups.	Groups with whom the dashboard is shared.

im viewdynamicgroup

viewdynamicgroup

`im viewdynamicgroup`

This view command can be used within the API to return detailed information on a selection of Integrity Lifecycle Manager dynamic groups.

Work Item

The dynamic group for which the information is returned.

Work Item Fields

The following table lists the fields added by the API for dynamic groups. For information on this command and its options, see the CLI man pages.

Field	Data Type	Description
`created`	Date	Date dynamic group was created
`createdBy`	User	User name that created the group
`lastModified`	Date	Date dynamic group last modified
`name`	String	Name of the dynamic group
`description`	String	Description of dynamic group.
`membership`	List of group and user items	List of group and user item field values for members of the dynamic group.

im viewfield

viewfield

`im viewfield`

This view command can be used within the API to return detailed information on a selection of Integrity Lifecycle Manager fields.

For more information, see `im fields` on page 127.

im viewgroup

viewgroup

`im viewgroup`

This view command can be used within the API to return detailed information on a selection of Integrity Lifecycle Manager groups.

For more information, see `im groups` on page 127.

im viewissue

viewissue

`im viewissue`

This view command can be used within the API to return a list of Integrity Lifecycle Manager items.

For more information, see `im issues` on page 128.

im viewproject

viewproject

`im viewproject`

This view command can be used within the API to return detailed information on a selection of Integrity Lifecycle Manager projects.

For more information, see `im queries` on page 131.

im viewquery

viewquery

`im viewquery`

This view command can be used within the API to return detailed information on a selection of Integrity Lifecycle Manager queries.

For more information, see `im queries` on page 131.

im viewreport

viewreport

`im viewreport`

This view command can be used within the API to return a list of created or shared Integrity Lifecycle Manager reports. It can also be used to return the properties of a specific report. You can specify which fields in the reports display. If you do not specify any fields for a list of reports, only the names of created and shared reports are returned. If you do not specify any fields for a specific report, all fields display.

Work Item

The reports for which the information is returned.

Work Item Context

The name of the user who created the report.

Work Item Fields

The following table lists the fields added by the API for the reports. These properties are editable from the CLI using the `im editreport` command. For information on all available fields, see the CLI man pages.

Field	Data Type	Description
`createdBy`	User	Name of the user who created the report.
`description`	String	Description of the report.
`id`	Integer	Database ID of the report. This is for PTC Technical Support only.
`isAdmin`	Boolean	Specifies the report as a system-provided object (objects within the Integrity Lifecycle Manager object model that support solution definition and management, as well as workflow migration).
`lastModified`	Date	Date the report was last modified.
`name`	String	Name of the report, and the user who created the report.
`query`	Query	Name of the query that defines the selection criteria for the report, and the user who created the query.
`references`	List of objects	All system provided and user objects that reference the report.
`shareWith`	List of users and groups	Users and groups that the report is shared with.
`sharedGroups`	List of groups.	Groups that the report is shared with.

Field	Data Type	Description
`reportTemplate`	String	Report template on which the report is based. For information on the report template format, see the *Integrity Lifecycle Manager Installation and Upgrading Guide*.
`recipeParams`	String	Report recipe parameters that define the report. For more information on the available parameters, see the knowledge base in the Support section of the PTC Web site (http://www.ptc.com).

im viewsegment

viewsegment

`im viewsegment`

This view command displays the segment (and optionally referenced or included subsegments), optionally as of a particular date.

Note

This command does not recurse if the segment is an ancestor tree. The segment is shown as an inserted segment, except if the `modelType` attribute is `im.Issue.subsegment.include`.

Work Item

The specified segment.

Work Item Fields

For information on the work item fields for this command, see the `im issues` on page 128 command. For information on all available fields, see the CLI man pages.

im viewstate

viewstate

`im viewstate`

This view command can be used within the API to return detailed information on a selection of Integrity Lifecycle Manager item types.

For more information, see `im states` on page 135.

im viewtype

viewtype

`im viewtype`

This view command can be used within the API to return detailed information on a selection of Integrity Lifecycle Manager item types.

For more information, see `im types` on page 135.

im viewuser

viewuser

`im viewuser`

This view command can be used within the API to return detailed information on a selection of Integrity Lifecycle Manager users.

For more information, see `im users` on page 136.

Configuration Management Commands

All published Configuration Management commands start with the command prefix `si`.

si about

about

`si about`

This view command can be used within the API to return product information.

Work Item

The application for which the product information is returned.

si activatedevpath

activatedevpath

`si activatedevpath`

Use this command to activate a development path that has been deactivated.

Work Item

The project or subproject to deactivate.

si add

add

`si add`

This action command can be used within the API to add one or more non-member files to a sandbox.

Note

Only an Integrity Lifecycle Manager client 2009 connecting to an Integrity Lifecycle Manager server 2009 properly displays status messages for all members. If you are using a different version of the Integrity Lifecycle Manager client or Integrity Lifecycle Manager server, display status messages are not guaranteed.

Work Item

The file to be added to the sandbox.

Work Item Context

The sandbox to which the file was added.

Work Item Result

Specifies the file that was added to the sandbox. The result can be one of the following:

- Member
- Destined member (for deferred additions)

Work Item Result Context

The project to which the file is being added. None for deferred additions.

si addmemberfromarchive

addmemberfromarchive

`si addmemberfromarchive`

Use this create command to add a member to a project using an existing archive from the repository.

Work Item

The member being added to the project.

Work Item Context

The project to which the member is being added.

Work Item Result

The project to which the archive is added.

Work Item Result Context

The project to which the archive is added.

si addsubproject

addsubproject

`si addsubproject`

This action command can be used within the API to add one or more subprojects to a configuration management project.

Work Item

The location of the subproject being added.

Work Item Context

The project to which the subproject was added.

Work Item Result

Specifies the path of the subproject that was added.

Work Item Result Context

The name of the project to which the subproject was added.

si addprojectattr

addprojectattr

`si addprojectattr`

Use this create command to add named attributes to a project. Only one attribute can be set per invocation. Requires use of the `--attr` option. An attribute is a `name=value` pair. An existing attribute with the same name is overwritten without prompt or error.

Work Item

The project to which the attribute is added.

Work Item Result

The attribute added to the project.

si archiveinfo

archiveinfo

`si archiveinfo`

This view command can be used within the API to return information on an archive.

Work item

The member for which archive information is returned.

Work Item Context

The project path.

Work Item Fields

The following table lists the fields added by the API for archives. For information on all available fields, see the CLI man pages.

Field	Data Type	Description
`MemberName`	String	Path and name of member archive.
`SandboxName`	String	Path and name of member's sandbox.
`ProjectName`	String	Path and name of the member's project.
`ArchiveName`	String	Path and name of archive.
`Archivetype`	String	Type of data stored in archive. This can be Text or Binary.
`ExclusiveLocking Mandatory`	Boolean	Specifies if exclusive locking is enforced on the archive.
`Compressed`	Boolean	Specifies if archive is

Field	Data Type	Description
		compressed.
`StorebyReference`	Boolean	Specifies if each revision is saved to a separate file.
`ArchiveDescription`	String	Description of archive.
`Labels`	List of label items	Revision labels in archive.
`Locks`	List of lock items	Locks in archive.

si checkpoint

checkpoint

`si checkpoint`

This action command can be used within the API to create a checkpoint of a project and add it to the project history.

Work item

The configuration path of the subproject or project to be checkpointed.

Work Item Context

The project location to which the subproject belongs.

Work Item Result

Specifies the new checkpoint.

si ci

ci

`si ci`

This action command can be used within the API to check in members of a sandbox.

Work item

The member that is checked in.

Work Item Context

Specifies that the change package was closed.

Work Item Result

Specifies the checked in revision.

The following table lists the potential field for the command result.

Field	Data Type	Description
`MemberRev`	Revision item	Revision number of member after checkin. Item ID is revision number.

si closecp

closecp

`si closecp`

This action command can be used within the API to close change packages.

Work item

The change package ID of the change package to be closed.

Work Item Result

Specifies that the change package was closed.

si co

co

`si co`

This action command can be used within the API to check out members into working files in a sandbox.

Work item

The member to be checked out.

Work Item Context

The project the file that is being checked out.

Work Item Result

Specifies the revision of the member who was checked out.

Field	Data Type	Description
`IsLocked`	Boolean	Specifies whether member is locked.

si configuresubproject

configuresubproject

`si configuresubproject`

This action command can be used within the API to configure a subproject in a configuration management project.

Work Item

The location of the subproject to be configured.

Work Item Context

The project location tow which the subproject belongs.

Work Item Result

Specifies the path of the configured subproject.

Work Item Result Context

The name of the project to which the subproject belongs.

si connect

connect

`si connect`

This action command can be used within the API to establish a connection to an Integrity Lifecycle Manager server.

si copypolicysection

copypolicysection

`si copypolicysection`

Use this action command to copy the policy settings from an existing policy section (global or project) to a new project policy section.

Work Item

The policy section to copy.

Work Item Result

Specifies the path of the configured subproject.

Work Item Result Context

The new project policy section to which the source policy section was copied.

si createcp

createcp

`si createcp`

Use this creation command within the API to create a change package for an Integrity Lifecycle Manager item.

Command Result

Specifies the change package ID of the change package that was created.

si createdevpath

createdevpath

`si createdevpath`

Use this creation command to create a new development path on a configuration management project revision.

Command Result

Specifies the variant project that was created.

The following table lists the potential fields for the command result.

Field	Data Type	Description
`ProjectName`	String	Name of the project.
`CanonicalPath`	String	Path of the project location.
`IsSubproject`	Boolean	Specifies if the project is a subproject.
`IsShared`	Boolean	Specifies if the project is a shared subproject.
`ParentProject`	String	Specifies the parent project of the variant project.
`IsVariant`	Boolean	Specifies if the development path is a variant project.
`DevelopmentPath`	String	Development path name.

Field	Data Type	Description
`IsBuild`	Boolean	Specifies if the project is a build project base don a specific checkpoint of the master project.
`BuildRevision`	Revision item	Revision number of the checkpoint that the build project is based on
`revisionId`	Revision item	ID of the project revision that was created.

si createproject

createproject

`si createproject`

Use this creation command within the API to create a new, empty project on the Integrity Lifecycle Manager server in a specified directory.

Command Result

Specifies the project that was created.

si createsubproject

createsubproject

`si createsubproject`

Use this creation command within the API to create a new subproject for a configuration management project.

Command Result

Specifies the subproject that was created.

Result Context

The project for which the subproject was created.

si deactivatedevpath

deactivatedevpath

`si deactivatedevpath`

Use this command to deactivate a development path. Later, you can reactivate the development path if necessary.

Note

This command can succeed with warnings. If there are errors or warnings, they appear under the affected work item.

Work Item

The project or subproject to deactivate.

si deletepolicysection

deletepolicysection

`si deletepolicysection`

This action command can be used within the API to delete a project policy section.

Command Result

Specifies the project policy section that was deleted.

si diff

diff

`si diff`

This view command can be used with the API to view differences between two revisions of a selection of members. It can also be used to view differences between the member revision and the working file.

Consult the CLI man pages for information on available options that control the operation of the difference algorithm and the member selections.

Note

The `--context` and `--guiCharacterEncoding` options cannot be specified for the API.

Work Item

Returns a boolean result if there are differences.

The following table lists the potential field for the command result.

Field	Data Type	Description
`Different`	Boolean	Specifies if there are differences.

si disconnect

disconnect

`si disconnect`

This action command can be used within the API to disconnect from the Integrity Lifecycle Manager server.

Command Result

Specifies the Integrity Lifecycle Manager server that was disconnected.

si drop

drop

`si drop`

Use this action command within the API to drop one or more member files or subprojects from a project.

Note

Only an Integrity Lifecycle Manager client 2009 connecting to an Integrity Lifecycle Manager server 2009 properly displays status messages for all members. If you are using a different version of the Integrity Lifecycle Manager client or Integrity Lifecycle Manager server, display status messages are not guaranteed.

Work Item

The work item can be one of the following:

- Member that was dropped
- Configuration path of the subproject that was dropped

The model type determines the type of work item.

Work Item Context

The context can be one of the following:

- For member work items, the full path of the project or subproject for the member.
- For former members, the full path of the sandbox for the member.

si dropprojectattr

dropprojectattr

`si dropprojectattr`

Use this action command to unset (drop) named attributes for a project. Only one attribute can be unset per invocation. Requires use of the `--attr` option. An attribute is a `name=value` pair. Attempting to unset a nonexistent attribute throws an `InvalidItemException` error.

Work Item

The project from which the attribute is being dropped.

Work Item Result

The attribute dropped from the project.

si dropsandbox

dropsandbox

`si dropsandbox`

Use this action command to unregister (drop) one or more existing sandboxes, optionally removing the working files and/or their directories.

Work Item

The sandbox selection.

Work Item Result

The sandbox selection dropped.

The following table lists the potential fields for the command result.

Field	Data Type	Description
`Working-Files-Deleted`	Boolean	Specifies if working files are deleted
`Sandbox-Directory-Deleted`	Boolean	Specifies if sandbox directory is deleted

si exit

exit

`si exit`

This action command can be used within the API to exit the current Integrity Lifecycle Manager client session.

si extenddevpath

extenddevpath

`si extenddevpath`

Use this action command to extend selected subprojects in an extendable development path the first time that changes to them are required.

Work Item

The subproject selection.

Work Item Context

The project or sandbox with the subprojects to extend.

Work Item Result

The extended variant subproject.

si freeze

freeze

`si freeze`

This action command can be used with the API to freeze one or more project members.

Work Item

The member selection.

Work Item Context

The project containing the member selection.

Work Item Result

The frozen revision.

si gui

gui

`si gui`

This action command can be used within the API to start the Integrity Lifecycle Manager client graphical user interface (GUI).

si locate

locate

`si locate`

This view command can be used within the API to return a list of locations where a project, subproject, member, or revision is used.

Work Item

Locations where a project, subproject, member, or revision is used.

Work Item Fields

The following table lists the fields added by the API for locate results. For information on all available fields, see the CLI man pages.

Field	Data Type	Description
`LowBuildRevision`	Revision item	First checkpoint to which the object belongs.
`HighBuildRevision`	Revision item	Last checkpoint to which the object belongs.
`WhereUsedItem`	Item	Object that has been located. The other fields returned by this command provide additional context for this object. If the located object is a build or variant project, additional fields are returned. For more information, see the Project Items table below.
`LowMemberRevi`	Revision	First revision for the member.

Field	Data Type	Description
sion	item	
HighMemberRevision	Revision item	Last revision for the member,
AddDate	Date	Date when the object was added.
DropDate	Date	Date when the object was dropped.

Project Items

The following table lists the fields added by the API for build or variant project items.

Field	Data Type	Description
Variant	Development path	Development path of the variant project.
Build	Revision item	Checkpoint on which the project is based.

si lock

lock

`si lock`

This action command can be used within the API to lock project members.

Work Item

The member that was locked.

Work Item Context

The project name for the file being locked.

Work Item Result

Identifies the locked revision.

The following table lists the potential field for the command result.

Field	Data Type	Description
BranchCreated	Boolean	Specifies if a branch was created for member that was locked.

si locks

locks

`si locks`

This view command can be used within the API to return information on a user's locks.

Work Item

The locked revision.

Work Item Context

The archive to which the locked revision belongs.

si movesandbox

movesandbox

`si movesandbox`

Use this action command to move a top-level sandbox to a new location.

Work Item

The sandbox that is being moved.

Work Item Result

The sandbox is moved.

si projectadd

projectadd

`si projectadd`

This action command can be used within the API to add an existing configuration management project to Integrity Lifecycle Manager.

Work Item

The project that is to be added

Work Item Context

Specifies the path of the project that was added.

si projectci

projectci

`si projectci`

This action command can be used within the API to check in members of a project.

Work Item

The member that is checked in.

Work Item Context

The project into which the file is being checked.

Work Item Result

Specifies the checked in revision.

The following table lists the potential field for the command result.

Field	Data Type	Description
`MemberRev`	Revision item	Revision number of member after checkin. Item ID is revision number.

si projectco

projectco

`si projectco`

This action command can be used within the API to check out members into working files.

Work Item

The member to be checked out.

Work Item Context

The project from which the file is being checked out.

Work Item Result

Specifies the revision of the member who was checked out.

The following table lists the potential field for the command result.

Field	Data Type	Description
`IsLocked`	Boolean	Specifies if member is locked.

si projectcpdiff

projectcpdiff

`si projectcpdiff`

This view command can be used within the API to return a list of the operations that occurred on a project between two checkpoints or timestamps if those operations were tracked in change packages.

Work Item

The command returns the set of differences for a specified range on a project. A list of change package entries represents the set of differences.

Work Item Fields

The following table lists the fields added by the API for projects. For information on all available fields, see the CLI man pages.

Field	Data Type	Description
`CPEntries`	List of entry items	Returns a list of items with change package entries. The fields include those from the `viewcp` command, with the following additions: • `id` contains the change package ID reference field. • `summary` contains the **Summary** field from the change package. • `user` contains the **User** field from the change package.
`diff-end`	String	Returns a reference that indicates the end revision diff.
`diff-start`	String	Returns a reference that indicates the start revision diff.
`recurse`	Boolean	Indicates if the command was called using `recurse`. It also indicates whether to expect some recursive entries in the output.
`MoveWorking File`	Boolean	Specifies if the working file was moved.

si projectinfo

projectinfo

`si projectinfo`

This view command can be used within the API to return information for a project.

Work Item

The configuration path of the project for which the information is being returned.

Work Item Fields

The following table lists the fields added by the API for projects. For information on all available fields, see the CLI man pages.

Field	Data Type	Description
`projectName`	String	Name of project.
`sharedFrom`		Returns a reference that indicates the end revision diff.
`projectType`	String	Returns a reference that indicates the start revision diff.
`server`	String	Indicates if the command was called using `recurse`. It also indicates whether to expect some recursive entries in the output.
`serverPort`	String	Port of server on which the projects reside.
`developmentPath`	Development path item	Development path name. Item ID is development path label.
`fullConfigSyntax`	String	Configuration path of project.
`revision`	Item	Revision number of project checkpoint. Item ID is revision number.
`mergedFrom`	Item	Indicates the development path merged into the parent project revision.
`manuallyMergedFrom`	List of revision items	Displays the source project revisions which were manually merged.
`lastCheckpoint`	Date	Date and time that the

Field	Data Type	Description
		checkpoint represents (not the creation date of the checkpoint).
`numMembers`	Integer	Number of members in project.
`numSubprojects`	Integer	Number of subprojects in project.
`description`	String	Detailed description of project.
`checkpointDescription`	String	Description of project checkpoint.
`projectAttributes`	List of attribute items	Attributes of project.
`developmentPaths`	List of development path items	Development path of project.
`ACLName`	String	Name of access control list (ACL) for project.
`ACLExists`	Boolean	Specifies if project ACL exists.
`actualACL`	String	Name of real ACL.
`associatedIssues`	List of issue items	Integrity Lifecycle Manager items associated with the project.

si projectlocks

projectlocks

`si projectlocks`

This view command can be used within the API to return information on locks by all users on all members of a sandbox or project.

Work Item

The locked revision.

Work Item Context

The archive to which the locked revision belongs.

si projects

projects

`si projects`

This view command can be used within the API to return a list of projects registered on the currently connected Integrity Lifecycle Manager server.

Work Item

The project configuration path.

Work Item Context

The parent project configuration path.

Work Item Fields

The following table lists the fields added by the API for projects.

Field	Data Type	Description
`ProjectName`	String	Name of project.
`IsShared`	Boolean	Specifies if the project is a shared subproject.
`IsSubproject`	Boolean	Specifies if the project is a subproject.
`ParentProject`	String	Name of parent project.
`CanonicalPath`	String	Path of the project location.
`IsVariant`	Boolean	Specifies if the project is a variant project based on a specific development path.
`Development Path`	String	Development path name.
`IsBuild`	Boolean	Specifies if the project is a build project based on a specific checkpoint of the master project.
`BuildRevision`	Revision item	Revision number of the checkpoint on which the build project is based.

si rename

rename

`si rename`

The member to be renamed. This is the old name of the member, before the rename operation.

Work Item

The project configuration path.

Work Item Context

The project path of the file.

Work Item Result

Specifies the new name of the renamed member.

Work Item Result Context

The project path of the file.

Result Fields

The following table lists the potential field for the work item result.

Field	Data Type	Description
`RenameWorking File`	Boolean	Specifies if working file for member was also renamed.

si restrictproject

restrictproject

`si restrictproject`

This action command can be used to restrict modifications to a live project configuration tree.

Work Item

The individual project configuration being restricted.

Work Item Context

Specifies the individual project configuration restricted.

The subprojects within the tree can be skipped due to its configuration or because it is already being restricted by another user. The following table lists the additional fields on the work item result.

Field	Data Type	Description
`isRestricted`	Boolean	Indicates whether the project configuration is restricted or not.
`isUpdated`	Boolean	Indicates whether this command changed the restriction on the given project configuration.

si resync

resync

```
si resync
```

Use this action command within the API to update a sandbox with the member revision.

Work Item

The work item can be one of the following:

- Resynced member
- Subsandbox that was updated
- Former member who was resynced

The model type determines the type of work item.

Work Item Context

The work item context can be one of the following:

- For members, the relative subproject the member is in or the full path to the sandbox if it is in the top-level sandbox.
- For former members, the absolute path in the sandbox for the member.

Work Item Result

Specifies the revision that was checked out if needed.

si revert

revert

```
si revert
```

Use this action command within the API to overwrite a sandbox file with a fresh copy of the working file, discarding changes.

Work Item

The work item can be one of the following:

- Member that was reverted
- Former member that was reverted

Work Item Context

The work item context can be one of the following:

- For members, the relative subproject the member is in or the full path name to the project if it is in the top-level project.
- For former members, the absolute path in the sandbox for the member.

Work Item Result

Specifies the working file that overwrote the sandbox file.

The following table lists the potential fields for the command result.

Field	Data Type	Description
`DeletedWorkingFile`	Boolean	Specifies if working file was removed.
`Unlocked`	Boolean	Specifies if working revision was unlocked.
`RemovedAdd`	Boolean	Specifies if add operations were removed.
`RemovedCheckin`	Boolean	Specifies if checkin operations were removed.
`RemovedDrop`	Boolean	Specifies if drop operations were removed.
`RemovedImport`	Boolean	Specifies if import operations were removed.
`RemovedRename`	Boolean	Specifies if rename operations were removed.
`RemovedUpdateRevision`	Boolean	Specifies if update revision operations were removed.
`RemovedMoveMember`	Boolean	Specifies if move operations were removed.

si rlog

rlog

```
si rlog
```

Use this view command with the API to return information about the member revision log.

Note

When using the `--fields` option in the API, unlike in the CLI, the `alllabels` field displays the same output as the labels field.

Work Item

The member selection.

Work Item Context

The project to which the member belongs.

Work Item Fields

The following table lists the fields added by the API for the member revision log. For information on all available fields, see the CLI man pages.

Field	Data Type	Description
`Revisions`	List of revision items	Specifies the member revisions.
`Date`	Datetime	Lock date.
`Author`	Author item	User name.

si sandboxes

sandboxes

`si sandboxes`

Use this view command to return information about sandboxes registered with the Integrity Lifecycle Manager client.

Work Item

The registered sandbox.

Work Item Context

The project the sandbox references.

Work Item Fields

The following table lists the fields added by the API for sandboxes. For information on all available fields, see the CLI man pages.

Field	Data Type	Description
`SandboxName`	String	Name of sandbox.
`ProjectName`	String	Name of project.
`ParentSandbox`	String	Name of parent sandbox.
`IsSubsandbox`	Boolean	Specifies if sandbox is a subsandbox.
`Server`	String	Server on which the project resides.
`Port`	String	Port of server on which the project resides.
`DevelopmentPath`	String	Development path name.
`BuildRevision`	Revision item	Revision number of the checkpoint that the build project is based on.

si sandboxinfo

sandboxinfo

`si sandboxinfo`

Use this view command within the API to return information about a sandbox.

Work Item

The sandbox for which the information is returned.

Work Item Context

The project the sandbox references.

Work Item Fields

The following table lists the fields added by the API for sandboxes. For information on all available fields, see the CLI man pages.

Field	Data Type	Description
`SandboxName`	String	Name of sandbox.
`ProjectName`	String	Name of project.
`ProjectType`	String	Type of project: normal, variant, or build.
`Server`	String	Server that project resides on.
`ServerPort`	String	Port of server that project resides on.

Field	Data Type	Description
`DevelopmentPath`	Development path item	Development path of sandbox. Item ID is development path label.
`FullConfigSyntax`	String	Configuration path of project
`Revision`	Item	Revision number of project. Item ID is the revision number.
`LastCheckpoint`	Date	Date and time project was last. checkpointed
`NumMembers`	Integer	Number of members in project.
`NumSubprojects`	Integer	Number of subprojects in project.
`Description`	String	Detailed description of project.
`Sparse`	Boolean	Specifies if sandbox is sparse.
`Shared`	Boolean	Specifies if sandbox is shared.
`ProjectAttributes`	List of attribute items	Attributes of project.
`SandboxAttributes`	List of attribute items	Attributes of sandbox.
`LineTerminator`	String	Line terminator used for sandbox.
`AssociatedIssues`	List of issue items	Integrity Lifecycle Manager items associated with the project.

si servers

servers

`si servers`

Use this action command with the API to set policy settings for a new or existing policy section (global or project).

Work Item

The policy section to set.

Work Item Result

The policy options in the specified policy section.

si setpolicysection

setpolicysection

`si setpolicysection`

Use this action command with the API to set policy settings for a new or existing policy section (global or project).

Work Item

The policy section to set.

Work Item Result

The policy options in the specified policy section.

si thaw

thaw

`si thaw`

Use this action command with the API to thaw one or more project members that are frozen. For more information, see freeze on page 157.

Work Item

The member selection.

Work Item Context

The project containing the member selection.

Work Item Result

The thawed revision.

si unlock

unlock

`si unlock`

Use this action command within the API to unlock a member.

Work Item

The member that was unlocked.

Work Item Context

The project name.

Work Item Result

Specifies the unlocked revision.

The following table lists the potential field for the command result.

Field	Data Type	Description
`BrokenLock Holder`	User item	User whose lock was removed.

si unrestrictproject

unrestrictproject

`si unrestrictproject`

This action command can be used to unrestrict a restricted project configuration tree.

Work Item

The individual project configuration being unrestricted.

Work Item Context

Specifies the individual project configuration unrestricted.

The subprojects within the tree can be skipped due to its configuration because it is already being restricted by another user. The following table lists the additional fields on the work item result.

Field	Data Type	Description
`isRestricted`	Boolean	Indicates whether the project configuration is restricted.
`isUpdated`	Boolean	Indicates whether this command changed the restriction on the given project configuration.

si updaterevision

updaterevision

`si updaterevision`

Use this command to update the member revision of the selection on a target project.

Work Item

The member selection.

Work Item Context

The project to which the member belongs.

Work Item Result

Specifies the updated member revision.

The following table lists the potential fields for the command result.

Field	Data Type	Description
`Deferred`	Boolean	Specifies if the operation is deferred.
`Previous-Revision`	Revision item	Specifies the previous member revision.

si viewcpentries

viewcpentries

`si viewcpentries`

This view command can be used within the API to return the net effect of a set of Integrity Lifecycle Manager change packages. It can also be used to return the net effect of all change packages on a selection of Integrity Lifecycle Manager items.

Work Item

The change package entry for which the information is returned.

Work Item Fields

In addition to the fields as specified by the `--attributes` and `--entryAttributes` options, the API adds the following field for organizing the list of entry items.

For information on all available fields, see the CLI man pages.

Field	Data Type	Description
`Type`	Item	Data type of the change package entry. It describes the following: • Action performed on the change package entry • Previous state, such as the previous revision for an update • Current state, such as committed or pending
`Membertype`	String	Describes the entry type, such as **Member** or **Subproject**.
`Member`	Item	Item describing the member or subproject.
`Revision`	Item	Member revision number.
`User`	Item	The **User** field from the change package.
`Timestamp`	DataType	The date and time, displaying in the format yyyy-ddThh:mm:ss. For example: 2015-03-20T19:44:39.
`Project`	Item	Project of the change package entry.
`Configpath`	Item	Project configuration path.
`Location`	String	Archive location of the change package entry.
`Variant`	Item	Name of the development path that the changes are on, if applicable.
`ID`	Item	Change package ID to which the change package entry belongs.
`Summary`	String	**Summary** field from the change package
`Server`	Item	Server hosting the change package.
`LinesAdded`	Long	Number of lines added to the member by the change package entry.
`LinesDeleted`	Long	Number of lines deleted from the member by the change package entry.
`BytesAdded`	Long	Number of bytes added to the member by the change package entry.

Field	Data Type	Description
`BytesDeleted`	Long	Number of bytes deleted from the member by the change package entry.
`IsText`	Boolean	Specifies whether the member is a text file or binary file.

si viewhistory

viewhistory

`si viewhistory`

Use this view command within the API to return member history information.

Work Item

The member history for which the information is returned.

Work Item Context

The project to which the member belongs.

Work Item Fields

The following table lists the field added by the API for member histories. For information on all available fields, see the CLI man pages.

Field	Data Type	Description
`Revisions`	List of revision items	Member revisions. For information on revision fields, see the CLI man pages.

si viewpolicysection

viewpolicysection

`si viewpolicysection`

Use this view command within the API to return member history information.

Work Item

The member history for which the information is returned.

Work Item Context

The project to which the member belongs.

Work Item Fields

The following table lists the field added by the API for member histories. For information on all available fields, see the CLI man pages.

Field	Data Type	Description
`Revisions`	List of revision items	Member revisions. For information on revision fields, see the CLI man pages.

si viewpolicysections

viewpolicysections

`si viewpolicysections`

Use this view command within the API to display the existing policies (global or project) on an Integrity Lifecycle Manager server.

Work Item

The policy section to display.

Work Item Fields

The following table lists the field added by the API to organize the policy settings for the policy sections being viewed. For information on all available fields, see the CLI man pages.

Field	Data Type	Description
`policies`	List of strings	List of policy key-value pairs as a string in the format `policy key=value` or `policy key;attribute=value`. For information on policies, see the CLI man pages. This field is only included in each work item in cases where the `–showPolicies` option is used when the `LockRecord` field is output.

si viewproject

viewproject

`si viewproject`

Use this view command within the API to return the contents of a project and information on its members.

Work Item

The work item can be one of the following:

- Member for which information is returned.
- Configuration path of the subproject for which the information is returned.

Work Item Context

The context for a member work item is the project.

si viewsandbox

viewsandbox

`si viewsandbox`

Use this view command within the API to return the contents of a sandbox.

Work Item

The work item can be one of the following:

- Sandbox member for which information is returned
- Former member for which information is returned
- Subsandbox for which information is returned
- Former subsandbox for which information is returned

Work Item Context

The work item context can be one of the following:

- For a member or former member, the parent sandbox
- For a sandbox or former sandbox, no context

Work Item Fields

The following table lists the fields added by the API for sandboxes. For information on all available fields, see the CLI man pages.

Field	Data Type	Description
`MemberRevLocked ByMe`	Boolean	Specifies if the member revision is locked by the current user in the member's project and development path context. This field is only added when the `LockRecord` field is output.
`WorkingRevLocked ByMe`	Boolean	Specifies if the working revision is locked by the current user in the member's project and development path context. This field is only added when the `LockRecord` field is output.

Authorization Administration Commands

All published Authorization Administration commands start with the command prefix `aa`.

aa addaclentry

addaclentry

`aa addaclentry`

Use this create command to associate a set of permissions with a principal within the scope of an ACL.

Command Result

A list of ACL entries restricted to the ACL entry changed by the invocation.

The following table lists the potential fields for the command result.

Field	Data Type	Description
`entries`	List of ACL entries	ACL entries added by this command invocation.
`createAcl`	Boolean	Indicates if the resulting ACL is newly created.
`Permission`	AclPermission	Permission for the ACL.
`Permitted`	Boolean	Permission is enabled or disabled for ACL
`Principal`	User or Group Item	User or group for ACL

aa editacl

editacl

`aa editacl`

Use this command to enable or break the inheritance of an ACL.

Command Result

Specifies the ACL that was updated

The following table lists the potential fields for the command result.

Field	Data Type	Description
`inheritParentPermission`	Boolean	Indicates if the ACL inheritance was enabled.

aa groups

groups

`aa groups`

Use this command to report a list of groups connected to the target Integrity Lifecycle Manager server.

Tip

The command can provide a list of users belonging to the groups, using the `--members` option.

Work Item

Principal name selection.

Work Item Result

A list of principals.

Note

If the `--members` option is specified, then principals (group) contain a list of principals (users) as a field.

aa users

users

`aa users`

Use this command to report a list of users connected to the target Integrity Lifecycle Manager server.

Tip

You can use the `--groups` option to see the groups to which the listed users belong.

Work Item

Principal name selection.

Work Item Result

A list of principals.

Note

If the `--groups` option is specified, then principals (group) contain a list of principals (users) as a field.

aa viewacl

viewacl

`aa viewacl`

Use this view command to display an ACL specified using the `--acl` option. You can select only one ACL per invocation.

Work Item

The ACL selection.

Work Item Result

The specified ACL item details.

Work Item Fields

The following table lists the potential fields for the command result.

Field	Data Type	Description
`inheritParent Permission`	Boolean	Inheritance is enabled or disabled for ACL
`Entries`	List of ACL entries	List of entries for ACL
`Permission`	AclPermission	Permission for the ACL.
`Permitted`	Boolean	Permission is enabled or disabled for ACL
`Principal`	User or Group Item	User or group for ACL

Integrity Lifecycle Manager Server Commands

All published Integrity Lifecycle Manager server commands start with the command prefix `integrity`.

integrity fetchviewset

fetchviewset

`integrity fetchviewset`

Use this command to retrieve the available published ViewSets and copy them to a specified directory.

Work Item

The ViewSet for which the information is returned.

Work Item Result

The ViewSet copied to a file.

integrity viewsets

viewsets

`integrity viewsets`

Use this view command to display the ViewSets available on the Integrity Lifecycle Manager server.

Work Item

The ViewSet for which the information is returned.

Work Item Result

The specified ViewSet details.

The following table lists the potential fields for the command result.

Field	Data Type	Description
`publishedState`	string	Published state of the ViewSet.
`name`	string	Name of the ViewSet.
`creator`	string	In a published ViewSet, name of the user who published the ViewSet. In a personal or unpublished ViewSet, the name of the creator of the ViewSet.
`modifiedDate`	datetime	Date the ViewSet was last modified.
`description`	string	Description of the ViewSet as added by the creator.
`mandatory`	boolean	Specifies whether **Mandatory** is enabled or disabled for a ViewSet. The Integrity Lifecycle Manager client automatically returns mandatory ViewSets and updates them when new versions become available on the Integrity Lifecycle Manager server.
`customizable`	boolean	Specifies whether **Customizable** is enabled or disabled for a ViewSet. When enabled, users can change the contents of ViewSet menus and toolbars.
`filename`	string	Path to where the ViewSet resides.

Working File Commands

All published working file commands start with the command prefix `wf`.

wf changes

changes

`wf changes`

This command returns a list of files for which you can submit changes. This command takes the path to a folder inside a sandbox.

Work Item

The full path to the file. The model type is `si.File`.

Work Item Fields

The following table lists fields returned by this command:

Field	Data Type	Description
`status`	integer	A bit field, where each bit corresponds to a different status. For status values, see Status Values on page 185.
`cpid`	string	Either null or the ID of the change package that contains deferred operations on this file.

Note

If a filter setting excludes non-member files, this command returns no results.

wf execute

execute

`wf execute`

Use this command to run a `si` command that takes members or non-members. This command is useful when the integration knows the filenames in the sandbox directory instead of member names. This command takes all options of the `si` command it is running.

When the `wf execute` command runs a `si` command, any errors returned by the `si` command are returned at the end of the command run. Errors do not prevent the command from completing.

Work Item

The work item for the command executed. See the documentation for the command being run by the `execute` command.

wf fileinfo

fileinfo

`wf fileinfo`

This command to returns the status of a file. For example, this command can be used to determine if to display an icon, or to enable (or disable) certain menus. This command takes a selection, and the full path to the list of the files (one or more files).

Work Item

The full path to the file. The model type is `si.File`.

Work Item Fields

The following table lists fields returned by this command:

Field	Data Type	Description
`status`	integer	A bit field, where each bit corresponds to a different status. For status values, see Status Values on page 185. If a filter setting excludes non-member files, the value is 0.
`cpid`	string	Either null, or the ID of the change package that contains deferred operations on this file. If a filter setting excludes non-member files, null is returned.

Status Values

The following table lists possible values for the status field:

Number	Name	Description
0x000001	Add	Either a deferred add operation and a file that is not a member, or non-member (and not ignored) in a sandbox.
0x000002	Drop	Either a file that has a deferred drop operation, or member that was deleted manually from a sandbox folder.
0x000004	Modified	A file that is a member, and that file has been edited, and that change has not been submitted to the repository.
0x000008	Moved	A file that has had a deferred moved operation performed on it.
0x000010	Renamed	A file that has had a deferred renamed operation performed on it.
0x000020	Needs Merged	A member that was merged automatically, and that file had merge conflicts, and the user opted to mark the member to be merged later.
0x000100	Member	A file that exists in the repository on the server.
0x000200	Former Member	A member that was dropped remotely, but still exists in the sandbox.

Number	Name	Description
0x000400	Out of Sync	A member that has a new revision that can be obtained by an `si resync` operation.
0x000800	Locked	A file that has been locked by a current user in Integrity Lifecycle Manager.

wf folders

folders

`wf folders`

Use this command to return a list of all directories that correspond to top-level sandboxes. Note the following about this command:

- The command is useful as an alternative to the `si sandboxes` command, because it eliminates duplicates by ignoring multiple `.pj` files in the same directory tree.
- The command can only run from the client integration point.
- The command is useful for integrations that have to mark folders in which command can be run. For example, it is useful for commands that run on the contents of a sandbox.

Work Item

The folder path.

Constraints Commands

Constraints commands provide the ability to view information about all of the input requirements for fields on Integrity Lifecycle Manager items. For more information, see Viewing Input Requirement Information Using Constraints Commands on page 71.

im itemconstraints

itemconstraints

`im itemconstraints`

This constraints command returns constraint shapes that describe the input requirements for fields on Integrity Lifecycle Manager items when creating or editing that item. You can view constraints for the following:

- An explicit list of item IDs specified in the command
- An implicit selection of items retrieved from a query, using the `--query` or `--queryDefinition` command option
- A new item of a specific type, using the `--newItemType` command option. Use this option to view the constraints that are applicable when creating a new item of the specified type.

The constraints in the response for this command can be used to extract information such as the following:

- Required fields
- Read-only fields
- Allowed state transitions based on the current state

Note

The entire workflow is not expressed. The only allowed state transitions that are shown are those that are applicable to the current user on the current item at the current point in time.

- Allowed values for a field
- Whether a field is single-valued or multi-valued
- Data type of input and the range of allowed values
- Values that are constrained by another field

The fields for which constraints are described in the API response correspond to the Integrity Lifecycle Manager fields relevant for the creation or editing of the Integrity Lifecycle Manager item.

Standard Data Types

The following Java data types are used for fields:

Java Name	Data Type
`java.lang.Boolean`	boolean
`java.util.Date`	date/datetime
`java.lang.Double`	double
`java.lang.Float`	float
`java.lang.Integer`	integer

Java Name	Data Type
java.lang.Long	long integer
java.lang.String	string
[B	binary byte array (for example, images)
com.mks.api.response.Item	(Integrity Lifecycle Manager API object)
com.mks.api.response.ItemList	(Integrity Lifecycle Manager API object)
com.mks.api.response.ValueList	(Integrity Lifecycle Manager API object)

The following C data types are used for fields:

C Name	Data Type
NULL	
unsigned short	boolean
time_t	date/datetime
double	double
float	float
int	integer
long	long integer
char *	string
wchar_t *	string
mksItem	(Integrity Lifecycle Manager API struct)
mksItemList	(Integrity Lifecycle Manager API struct)
mksValueList	(Integrity Lifecycle Manager API struct)

Unicode Support for C API

The C API has been designed to be fully Unicode by using the `wchar_t *` data type. However, there is no Unicode support for C API commands that accept or return user names, passwords, host names, or file names. This includes the following functions:

- `mksCmdRunnerGetDefaultHostname`
- `mksCmdRunnerGetDefaultUsername`
- `mksCmdRunnerGetDefaultPassword`
- `mksCmdRunnerGetDefaultImpersonationUser`

- `mksCmdRunnerSetDefaultHostname`
- `mksCmdRunnerSetDefaultUsername`
- `mksCmdRunnerSetDefaultPassword`
- `mksCmdRunnerSetDefaultImpersonationUser`
- `mksCreateIntegrationPoint`
- `mksAPIInitialize`
- `mksCreateSession`
- `mksIntegrationPointGetHostname`
- `mksSessionGetUsername`
- `mksSessionGetPassword`

User Item Data Type

The ID of the user item data always contains the user ID field, which is the user login ID. It can also contain an additional `FullName` field, depending on the support for user full names from the Integrity Lifecycle Manager server. For more information on full name support, see the *Integrity Lifecycle Manager Installation and Upgrading Guide*.

Date and Datetime Data Types

Date and time fields display the time as midnight, based on the local time zone of the client that is running the API.

If you are running on a UNIX platform, you can set the time zone globally. You can use the *$TZ* environment variable to set the time zone locally.

6

Administering the Integrity Lifecycle Manager API

This chapter provides information on how to set up and administrate the Integrity Lifecycle Manager API.

It contains information on the following:

- Setting Up Permissions on page 192
- Setting Up Policies on page 192
- Setting Up Impersonations on page 193
- Patch Management on page 195
- Troubleshooting on page 195

Setting Up Permissions

To be able to run certain commands, all users of the integration must be set up to have permission to view Integrity Lifecycle Manager administrative information. The ability to view Integrity Lifecycle Manager administrative information is controlled through Access Control Lists (ACLs). For more information, see the *Integrity Lifecycle Manager Installation and Upgrading Guide*.

Setting Up Policies

Two policies control access to the Integrity Lifecycle Manager server and the Integrity Lifecycle Manager client:

- Connection policy
- Authentication policy

These policies are specified in the `IntegrityClient.rc` file on the Integrity Lifecycle Manager client (`.IntegrityClient.rc` for UNIX) and in the `config/IntegrityClientSite.rc` file on the Integrity Lifecycle Manager server. The files are located in your home directory on the client or the install directory of the server:

- `daemon.connectionPolicy` specifies the connection policy.
- `daemon.authenticationPolicy` specifies the authentication policy.

The settings for these policies depend on your integration point. Integrity Lifecycle Manager has set defaults for both the client and the server integration points.

Client Integration Point Policy Defaults

The following defaults are set for Windows clients.

- `mks.ic.common.policy.ICLocalConnectionPolicy`

  Allows connections from `localhost` only.
- `mks.ic.common.policy.ICNoAuthenticationPolicy`

  Allows everything to connect.

The following defaults are set for UNIX clients.

- `mks.ic.common.policy.ICAllowAllConnectionPolicy`

  Allows everything to connect.
- `mks.ic.common.policy.ICLocalAuthenticationPolicy`

  Validates the user through a local system file.

Server Integration Point Policy Defaults

The following defaults are set for servers.

- `mks.ic.common.policy.ICAllowSpecificConnectionPolicy`

  Allows a specific set of IP addresses to connect. You must specify the IP addresses in comma-delimited format for the `daemon.validConnectionList` property.

- `mks.ic.common.policy.ICAuthenticatedSessionPolicy`

  Validates the standard HTTP authorization credentials against an external URL. Each session can have a different set of credentials associated with it. Initially, a session calls the URL to make sure that the credentials are valid. On subsequent requests for that session, the credentials passed in on the request are compared to the previously stored validated credentials.

  By default, the Integrity Lifecycle Manager authentication servlet (`http://host:port/Authenticate`) is used to validate against the configured realms.

You can also use `ICAllowAllConnectionPolicy` as the connection policy for a server integration point. If you use this policy, specify the users who are allowed to connect in comma-delimited format for the `daemon.validUsersList` property. If you do not set this property, all authenticated users are allowed to connect.

Client as Server Integration Point Policies

No policy defaults are set for a client as server integration point. If you are using this scenario, the recommended defaults are the same as for a server integration scenario.

Setting Up Impersonations

The ability to impersonate and the users who can be impersonated are controlled through ACLs. For more information on ACLs, see the *Integrity Lifecycle Manager Installation and Upgrading Guide*. Currently, the impersonation ACLs can only be set up through the Authorization Administration Web interface or the CLI. They cannot be configured through the Integrity Lifecycle Manager client. For information on configuring an ACL using the Web interface, see the *Integrity Lifecycle Manager Installation and Upgrading Guide*.

For information on working with the Authorization Administration CLI, see the CLI man pages.

The format of the impersonation ACL can be one of the following:

- `mks:impersonate:user:<user name>`
- `mks:impersonate:group:<group name>`

The ACL name identifies the specific user or user group that can be impersonated. The principal of the ACL is the specific user or group. This user or group is either allowed or denied permission to impersonate the user group specified in the ACL name. For example, assume that an ACL of `mks:impersonate:user:bill` has the user group "Manager" listed as a principal and is allowed permission. Anyone in the Manager user group can impersonate Bill.

The following rules apply to impersonation ACLs:

- The impersonation permission is not available if there are no ACLs
- Individual user ACLs take precedence over group ACLs.
- Assume that a requesting user belongs to multiple groups. Some of these groups are allowed to impersonate a particular user or group. Some of these groups are explicitly denied the impersonation of the user or group. The requesting user is not allowed to impersonate the user or group.

For more information on using impersonation, see Using Impersonation on page 56.

Managing Server Connections for Impersonated Users

When impersonating a user, it results in multiple Integrity Lifecycle Manager client connections for the user who is performing the impersonation. To determine which server connection to close, the servers command displays a complete list of server connections. The `disconnect` command allows you to specify the server connection to disconnect.

For example, typing `im server` displays:

```
jriley@abcFinancial:7001
jriley@abcFinancial:7001 (nsingh)(default)
nsingh@abcFinancial:7001
```

Where:

- `jriley@abcFinancial:7001` is the regular server connection for `jriley` who wants to impersonate `nsingh`.
- `jriley@abcFinancial:7001 (nsingh)(default)` is `jriley`'s impersonated server connection.
- `nsingh@abcFinancial:7001` is the server connection for `nsingh` that `jriley` wants to impersonate.

Tip

You can also specify the `--gui/g` option to manage impersonated connections from the **Select Server Connection** window

To disconnect the impersonated server connection, type `im disconnect --impersonateuser=jriley`.

To disconnect the regular server connection for `jriley`, type `im disconnect --user=jriley`.

Note

If the impersonated server connection is not the default server connection, you must specify `im disconnect --user --impersonateuser` to disconnect the impersonated server connection.

Patch Management

When executing commands on an Integrity Lifecycle Manager client, the patch manager is disabled. No patches are checked/installed as result of API connections to a client. Although the command `updateclient` can be run through the API, it is not recommended since it requires interaction and often requires a restart of the client.

Troubleshooting

If you encounter a problem with the Integrity Lifecycle Manager API, contact PTC Technical Support. Before making contact, PTC recommends that you gather information as described in subsequent topics.

Using the API Viewer

You can use the API Viewer to re-create your problem and then send the results to PTC Technical Support. The results can help isolate API problems from integration code problems.

Locating Your API Version Number

To provide you with accurate assistance, your Technical Support Representative must know the following:

- Integrity Lifecycle Manager API version used by your integration
- Integrity Lifecycle Manager API versions supported by the versions of Integrity Lifecycle Manager communicating with your integration

Note

The API version is different than the Integrity Lifecycle Manager version.

The API version returned by Integrity Lifecycle Manager is the highest API version that it can support. For example, Integrity 10.4 includes Integrity API 4.12. This means that 4.12 is the highest version it supports when an integration communicates with it. However, it also supports API sessions for Integrity API 4.11 and 4.10 when requested.

Finding the API Version Number Used by your Integration

Examine the source code of the integration, specifically, the integration points and sessions.

Finding the API Version Number Supported by Integrity Lifecycle Manager

You can find the API version on the Integrity Lifecycle Manager home page. Your administrator can provide you with its location. The URL for the Integrity Lifecycle Manager client home page displays in the form:

```
http://servername:portnumber
```

An example follows:

```
http://intra-wif:7001
```

This information is also available in the following locations:

- **Help ▸ About** window of the Integrity Lifecycle Manager client GUI and Web interface
- `si/im/integrity/aa` about commands
- `mksapi.log` file

Using API Logging

By enabling API logging, you can capture information that helps your Technical Support Representative solve your problem

Enabling Logging for the Java API

Logging must be enabled before getting the first instance of `IntegrationPointFactory`. To enable API logging, use the `MKSLogger(logfile)` constructor. All log data is sent to the file specified by `logfile` or `system.out` if null is passed in.

For the Java API, different logging methods can be defined. For more information, see the Integrity Lifecycle Manager Java API documentation accessed through the Integrity Lifecycle Manager server Web page.

Enabling Logging for the C API

Calling the `mksAPIInitialize(char* logfile)` function initializes logging. Log data is sent to the log file specified. If the parameter is NULL, then the log data is sent to mksapi.log in the current working directory. Once the `mksAPIInitializ`e function is called, the logging behavior cannot be dynamically changed.

For the C API, different logging mechanisms can be defined. For more information, see the Integrity Lifecycle Manager ANSI C API documentation accessed through the Integrity Lifecycle Manager server Web page.